Titre: Title:	Évaluation des entrelaceurs au sein des Codes Turbo par simulations
Auteur: Author:	Grégory Bruno Ludevic Royer
Date:	2000
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Royer, G. B. L. (2000). Évaluation des entrelaceurs au sein des Codes Turbo par simulations [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8597/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/8597/
Directeurs de recherche: Advisors:	
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

Évaluation des entrelaceurs au sein des Codes Turbo par simulations

GRÉGORY ROYER
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES (M. Sc. A)
(GÉNIE ÉLECTRIQUE)
NOVEMBRE 2000

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

Évaluation des entrelaceurs au sein des Codes Turbo par simulations

présenté par : Grégory Royer

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

Professeur Jean Conan, Ph. D., Président

Professeur David Haccoun, Ph. D., Directeur de recherche

Professeur François Gagnon, Ph. D., Membre du jury

À MES PARENTS ET EMERIK

Remerciements

Le travail présenté dans ce mémoire n'aurait pu être mené à terme sans l'aide et le soutien de nombreuses personnes.

Je tiens tout d'abord à exprimer toute ma gratitude à mon directeur de recherche, le professeur David Haccoun, pour son soutien constant et sa confiance qu'il m'a accordée pendant mes années de maîtrise. Je tiens aussi à le remercier pour son soutien financier.

Je tiens surtout à remercier mes parents sans qui je n'aurais pas atteint un tel niveau d'études. Ils ont réussi à me soutenir moralement et à m'aider dans les moments les plus difficiles. Ils ont réussi à me montrer le bon chemin à chaque fois que je dérivais. Enfin, c'est grâce aussi à leur soutien financier que j'ai pu profiter de mes années universitaires. Ma gratitude va également à mon frère et Guylaine qui ont toujours été là quand il le fallait.

Je tiens à émettre aussi une pensée à tous mes collègues de la section de communications, Afif, Mehdi, Mélita, Guillaume et Pierre-Frédéric. J'ai énormément apprécié travailler en compagnie d'Afif.

Je n'oublierai pas tous mes amis, Julien, Hervé, Richard, Marguerite, Thierry, Anne, Joan et tous les autres. Je remercie en particulier Julien pour la lecture et les corrections pertinentes de ce mémoire

Résumé

Le présent mémoire porte sur l'évaluation des performances des Codes Turbo. Ces derniers font partie des techniques les plus puissantes pour la correction des erreurs pour les transmissions numériques. L'encodage se fait à l'aide d'une concaténation parallèle de deux codeurs convolutionnels récursifs et systématiques à travers un entrelaceur.

C'est par le décodage que ces codes sont qualifiés de turbo. Le décodage est itératif et s'effectue à l'aide de deux décodeurs en série. Une information de fiabilité (information extrinsèque) est transmise entre les deux décodeurs permettant ainsi au décodeur suivant de corriger des erreurs que le précédent n'aurait pu modifier. Pour ce processus, l'algorithme MAP est utilisé. Ce dernier permet d'atteindre de très bonnes performances proches de la limite de Shanon.

L'analyse des performances des Codes Turbo est très complexe. En effet, pour ce faire, il faut considérer tous les éléments de cet encodage. Nous pouvons penser à différents paramètres comme ceux des codeurs convolutionnels qui constituent l'encodage. Nous pouvons également penser à l'algorithme de décodage qui modifie de façon conséquente les performances.

Les performances des Codes Turbo en fonction des entrelaceurs font le sujet de ce mémoire. Dans un premier temps, les entrelaceurs pseudo-aléatoires sont considérés. L'évaluation des performances des entrelaceurs aléatoires purs montrent qu'ils sont d'excellents systèmes. L'entrelacement symétrique est introduit en y ajoutant la notion d'aléatoire : la notion de séparation de distance après entrelacement est utilisée. Ceci permet d'imposer une séparation minimum entre les bits consécutifs après entrelacement. Les performances de cet entrelacement hybride sont meilleures que celles de l'aléatoire. Toutefois, une analyse de la complexité de l'entrelaceur symétrique S révèle qu'il est parfois difficile de générer un tel processus

Les entrelaceurs déterministes sont ensuite étudiés. Ils sont caractérisés par la connaissance à l'avance de la position des bits après entrelacement. L'entrelacement bloc est le premier étudié et montre de bonnes performances pour de petites tailles de bloc. Néanmoins, à mesure que la taille des blocs augmente, ces performances se détériorent. La complexité de cet entrelacement est moindre. L'entrelacement convolutionnel est ensuite considéré. Les performances obtenues avec ce type d'entrelacement ne sont pas excellentes, quelle que soit la taille de bloc utilisée.

Enfin, l'entrelacement de type bloc-hélicoïdal est présenté. Ce dernier présente des caractéristiques bien particulières car il permet d'obliger une certaine séparation entre les bits après entrelacement. De plus, sa simplicité est excellente car le processus inverse de l'entrelaceur est simple grâce à des symboles de synchronisation. Trois types d'entrelacement bloc-hélicoïdal sont présentés et les trois donnent de bonnes performances quelle que soit la taille de bloc considérée. Toutefois, le comportement est meilleur pour des faibles tailles de bloc.

Abstract

This thesis evaluates the performances of Turbo Codes. These are among the best error correcting codes for digital communications. They are concatenated codes composed of two recursive and systematic convolutionnal codes. An interleaver is used to allow the two encoders to see different sequences of bits.

The decoding process is what gives to those codes the property of turbo. This process is iterative and is done with two decoders in serie. The extrinsec information links the two decoders and offers some reliability information for the second decoder. This allows the second decoder to correct errors the first one may have not modified. The MAP algorithm is used and allows the Turbo Codes to reach excellent performances.

Many factors have to be considered to analyze the performances of Turbo Codes. The parameters of the two convolutionnal encoders are among them. We may also consider the decoding algorithm.

Nevertheless, the interleavers are the sub-systems considered in this thesis. The pseudo-random interleavers are first studied. The pure random interleaver is evaluated and gives very good performances. We then introduce the symetric interleaver, to which we add the property of randomness. This allows to induce a separation between two consecutive bits after interleaving. Those interleavers give much better performances than the pure random ones. Nevertheless, the complexity may be high depending on the chosen parameters.

The deterministic interleavers are classical and of low complexity. The block interleaver is evaluated and gives very good performances for short block sizes. However, the performances decrease with the block size. The convolutional interleavers are more

complex because they induce some delay to the encoding process. Their performances are not excellent, wether the block size is large or not.

Finally, the block-helical interleaving process is introduced. It allows a certain separation after interleaving and its deinterleaving process is easy thanks to synchronization bits. There are three types of these interleavers, which gives same kind of good performances. Nevertheless, the behavior of such an interleaver seems better for short block sizes.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1 - Introduction

1.1 Théorie de l'information

Un système de communications numériques est un moyen de transport de l'information entre un utilisateur et un autre. Le terme numérique est relatif à la séquence de symboles qu'il utilise pour représenter l'information. En général, pour les communications numériques l'alphabet utilisé est binaire. Ce genre de transmission de données est très intéressant dans le sens où il permet l'utilisation de nombreuses techniques de manipulation de l'information. Parmi celles-ci, nous pouvons penser à la compression de l'information, à l'encryptage de cette dernière, mais aussi aux codes correcteurs d'erreurs. Ce sont ces derniers qui font l'objet de beaucoup de recherches actuellement.

En 1948, Shannon a publié un article intitulé "A Mathematical Theory of communication" dans le Bell Systems Technical Journal [48]. Par la suite, il publia un second article en 1949, toujours dans le même journal "Communication Theory of Secrecy Systems" [49]. Ces deux articles faisaient référence au travail sur le codage de l'information. Le premier de ces deux papiers définit la manière dont nous effectuons des transmissions numériques de nos jours. Il s'agit de la base de la théorie de l'information. Dans cet article, il introduit la métrique au travers de laquelle l'information peut être quantifiée. Cette métrique définit le nombre minimum de symboles nécessaires de façon à numériser un message sans faire d'erreur. À partir de ceci, un message qui contient la même information mais un nombre de symboles supérieur est redondant et ouvre la porte au codage de l'information et à la correction des erreurs grâce aux codes. C'est ainsi que trois types de codages ont pu être introduits :

- codage de source : ce dernier est utilisé pour supprimer la redondance dans les paquets d'information de la source;

- codage d'encryptage : seul le destinataire peut décoder l'information transmise grâce à ce type d'encodage;
- codage correcteur d'erreur : un tel codage sert à minimiser l'impact du bruit. Ceci est réalisé en introduisant de la redondance. Ce type de codage est aussi appelé codage de canal.

Finalement, il est important de relater le théorème de Shannon à la base des codes correcteurs d'erreur. Pour tout canal, il est possible d'associer une capacité de canal C. Il existe alors des codes de contrôle d'erreur de façon à ce que l'information puisse être transmise au travers du canal à des taux inférieurs à C avec une probabilité d'erreur par bits faible.

Shannon fut un des précurseurs des codes correcteurs d'erreur. Il y en a eu bien d'autres. Un autre nom que nous pouvons mentionner dans ce chapitre d'introduction est Hamming. Ce dernier était un contemporain de Shannon et travaillait pour les laboratoires Bell dans les années quarante. Beaucoup le considèrent comme le fondateur du sujet des codes correcteurs d'erreur grâce à son article "Error Detecting and Error Correcting Codes" paru dans le Bell System Technical Journal en 1950 [24]. Ce grand chercheur est très connu pour ses fameux codes de Hamming.

1.2 Les premiers usages des codes correcteurs d'erreur

C'est le domaine spatial qui a permis le développement de ces codes correcteurs d'erreur. En effet, les premières utilisations de ceux-ci en communication devaient permettre de surmonter la perte de puissance lors de la propagation entre le récepteur ou transmetteur de l'engin spatial et le transmetteur ou le récepteur de la station terrestre. Les engins envoyés dans l'espace, à l'origine, se devaient d'être assez petits pour des raisons économiques, mais aussi des raisons d'énergie, comme la poussée des moteurs. Par

conséquent, emporter une grande antenne, ou de grosses batteries, ou encore de grands panneaux solaires dans ces appareils était hors de question. Réduire la taille de l'antenne entraînait obligatoirement une perte de puissance à l'émission ou à la réception. Le signal transmis avait donc une puissance moindre. Néanmoins, le bruit est toujours le même. Ainsi, de nombreuses erreurs pouvaient facilement détériorer le signal.

L'idée des codes correcteur d'erreur est de gagner en terme de puissance (dB). En général, nous parlons de rapport signal sur bruit. Le gain de codage est la différence entre un rapport signal à bruit E_b/N_0 requis pour atteindre une certaine performance d'erreur par bit (BER) avec un système codé et le E_b/N_0 requis pour atteindre la même performance d'erreur avec un système non codé.

De façon plus spécifique, nous allons maintenant parler du choix de la modulation. En ce qui concerne les transmissions spatiales, la modulation binaire de phase BPSK est la plus sensée car elle offre une meilleure puissance par rapport aux autres modulations. Le gain de cette modulation par rapport à la modulation binaire de fréquence BFSK est de 3 dB [46] [52]. Une transmission BPSK non codée nécessite un E_b/N_0 de 9.6 dB pour atteindre une performance de 10^{-5}. En utilisant un codage de Reed-Solomon avec un code convolutionnel, de façon standard, il est possible d'atteindre la même performance à 2.2 dB. Ceci équivaut donc à un gain de codage de 9.6-2.2, soit 7.6 dB.

Nous venons de voir que les codes correcteurs d'erreur permettent de réduire la valeur du E_b/N_0 nécessaire à l'entrée du démodulateur pour obtenir la même performance d'erreur sans codage. Ceci a un impact notamment sur les exigences requises vis-à-vis des paramètres de liaison. Grâce au codage, des communications seront fiables pour de plus grandes distances étant donné que la puissance d'émission peut être réduite. Il est également possible de réduire la taille des antennes, ce qui a, comme nous l'avons vu, un impact matériel et économique sur l'envoi d'engins spatiaux entre autres. Un gain de 1 dB est équivalent à une grande économie d'argent pour les concepteurs de systèmes de

communication. Actuellement, on évalue à 80 millions de dollars le gain de 1 dB [52], mais ceci n'est qu'une estimation.

Avec de tels chiffres, il serait possible de penser réduire au maximum le coût d'un appareil de transmissions numériques. Malheureusement, il en est tout autrement dans la vie de tous les jours. En effet, il existe une limite au codage. Shannon a ainsi défini la capacité de canal en considérant le cas du bruit blanc additif et gaussien (AWGN) comme :

$$C = W \log_2\left(1 + \frac{S}{N}\right) \text{ bits par seconde} \qquad (1.1)$$

Dans cette équation, W est la largeur de bande du canal en Hertz, S est la puissance moyenne du signal et N la puissance du bruit dans le canal. À partir de cette équation, il est possible de développer une limite sur le gain de codage. L'efficacité spectrale η est introduite en bits par seconde par Hertz. Cette dernière représente le nombre moyen de bits d'information transmis par intervalle de signalisation. La valeur de E_b/N_0 est alors :

$$\frac{E_b}{N_0} \phi \frac{2^{\eta} - 1}{\eta} \qquad (1.2)$$

Jusqu'au début des années 90, les chercheurs ont été capables d'aller gagner jusqu'à 2 dB de cette limite. Néanmoins, il restait tout de même ces 2 dB à aller chercher. De nouveaux codes ont permis d'approcher d'avantage cette limite, à savoir les Codes Turbo qui ne sont qu'à 0.3 dB de la capacité.

1.3 L'avènement des Codes Turbo

Les codes correcteurs d'erreur peuvent être classifiés en deux catégories, à savoir les codes blocs et les codes convolutionnels. Tandis que les premiers requièrent la subdivision au préalable des bits d'information en paquets ou blocs d'information, les seconds sont plus intéressants puisqu'ils permettent un codage de l'information de façon continue.

Les codes convolutionnels sont parmi les plus utilisés dans les communications sans fil de par leur propriété de continuité dans la transmission de l'information. En effet, nous n'avons pas à attendre que le bloc de bits soit arrivé avant de transmettre. Leur performance est excellente et ceci grâce à la puissance des algorithmes de décodage disponibles. Entre autres, nous pouvons penser au décodage de Viterbi qui fut une révolution. Cet algorithme est à maximum de vraisemblance et donc optimal; il assure le minimum de probabilité d'erreur par séquence. Néanmoins, ces codes ont un désavantage de taille. En augmentant la longueur de contrainte, nous augmentons certes le gain de codage, mais également la complexité du décodage de façon exponentielle. Ceci est tel que les systèmes pratiques sont limités à une longueur de contrainte égale à 9

Les codes convolutionnels connaissant des limites, la recherche s'est concentrée sur la conception de codes correcteurs d'erreur plus puissants. L'idée de base était évidemment de conserver l'avantage des codes convolutionnels tout en se débarrassant de leur contrainte. De multiples tentatives ont aboutit aux codes produit [18], aux codes concaténés [20], aux codes multi-niveaux [31], mais aussi aux algorithmes sous-optimaux pour le décodage des codes convolutionnels avec de grandes longueurs de contraintes, comme le décodage séquentiel [10], le décodage de Viterbi adaptatif [13] et le décodage bidirectionnel [11].

En 1993, Berrou Glavieux et Thitimajshima ont développé une nouvelle approche aux codes correcteurs d'erreur : les Codes Turbo [8]. Il s'agit en fait de la concaténation parallèle de deux codeurs convolutionnels récursifs et sytématiques de faible longueur de contrainte à travers un entrelaceur. Le décodage est effectué de façon itérative par deux décodeurs concaténés en série. Les auteurs cités ci-dessus ont montré que nous ne sommes plus qu'à 0.7 dB de la borne de Shannon. Étant donné qu'avec les codes correcteurs d'avant, nous en étions à 2.2 dB, ceci veut dire que l'avènement des Codes Turbo apportait un gain de codage additionnel de 1.5 dB. Actuellement, les Codes Turbo se situent à 0.2 dB de cette fameuse limite [38].

Les performances des Codes Turbo sont fonctions non seulement des codes convolutionnels utilisés, mais aussi de l'algorithme de décodage et des entrelaceurs. De nombreuses recherches ont porté sur les deux derniers points. Les Codes Turbo étant très complexes, une des méthodes utilisées pour leur étude est d'examiner l'effet de chacun des paramètres un à la fois. Dans ce mémoire, nous nous sommes concentrés sur l'effet des entrelaceurs sur la performance des Codes Turbo. Ces derniers font l'objet de beaucoup de recherche et de nombreux algorithmes ont été développés de façon à améliorer les performances des transmissions numériques. Une des utilités des entrelaceurs est de mélanger l'information de façon à annuler l'effet néfaste des canaux de transmission. Pour les Codes Turbo, ces entrelaceurs permettent surtout au décodeur de pouvoir corriger les erreurs de façon subséquente. Nous nous intéresserons à l'effet des entrelaceurs dans ce mémoire.

1.4 Composition du mémoire

Pour l'étude des entrelaceurs au sein des Codes Turbo, nous avons divisé notre recherche en plusieurs parties. Les premiers chapitres de ce mémoire portent sur l'aspect théorique des systèmes de communication et des Codes Turbo. Ainsi, dans le second chapitre, nous nous intéressons à ce que nous appelons la concaténation parallèle. Cette dernière est une

méthode utilisée de nos jours pour le codage de l'information que nous devons corriger à la réception. Il s'agit d'utiliser deux codeurs en parallèle qui reçoivent la même séquence d'information, mais dans un ordre différent. Dans ce chapitre, nous introduisons tout d'abord les systèmes de communication. Le codage convolutionnel qui est la base des Codes Turbo est aussi introduit de façon théorique. Nous en verrons deux types, à savoir le codage convolutionnel non récursif et le récursif. Par la suite, la concaténation parallèle est vue en détail et nous terminons ce chapitre par différents paramètres pertinents aux transmissions numériques : la modulation BPSK et les différents types de canaux disponibles.

Le troisième chapitre fait l'objet d'un des paramètres importants des Codes Turbo. En effet, sans l'algorithme de décodage, ces derniers ne sont plus d'excellents correcteurs d'erreur. Il existe de nombreux algorithmes de décodage et un des meilleurs pour les Codes Turbo s'avère être l'algorithme MAP. L'étude qui en sera faite sera de nature théorique. Nous voyons en particulier tous les paramètres nécessaires à cet algorithme comme les métriques d'états. Nous restreignons notre étude au canal AWGN et enfin présentons quelques variantes de cet algorithme.

L'algorithme de décodage étant présenté, nous pouvons nous intéresser aux Codes Turbo. Ces derniers font partie du chapitre suivant. Le principe de décodage itératif est présenté en premier lieu car il est à la base du décodage de ces codes. Les différents termes pertinents de l'information extrinsèque sont développés. Ces derniers sont nécessaires au bon fonctionnement du décodage. Enfin, le décodage turbo est introduit. Par la suite, dans ce chapitre, l'élément que nous allons étudier dans tout le reste du mémoire est présenté. Il s'agit de l'entrelaceur. Une grande partie de la théorie des entrelaceurs est faite à la fin du chapitre. Nous verrons, entre autres, le facteur d'étalement qui est un paramètre pertinent à l'étude des Codes Turbo.

Une fois les entrelaceurs introduits, il est possible de commencer l'étude qui fait l'objet de ce mémoire. en effet, dans le chapitre 5, les entrelaceurs pseudo-aléatoires sont étudiés. Tout d'abord, l'intérêt se porte sur les entrelaceurs purement aléatoires. Ces derniers nous servent de tremplin pour la description des performances des Codes Turbo en général. Par la suite, les entrelaceurs symétriques sont introduits pour enfin nous concentrer aux entrelaceurs symétriques S dont l'intérêt porte sur l'aspect d'aléatoire et de symétrie.

Le chapitre suivant porte sur des entrelaceurs plus classiques qui sont déterministes. Il s'agit tout d'abord des entrelaceurs blocs dont la simplicité est notoire. Les entrelaceurs convolutionnels font alors l'objet d'une étude assez poussée pour en comprendre le fonctionnement au sein des Codes Turbo. Ces deux types d'entrelaceur ont en commun la connaissance de la position des bits de sortie à l'avance.

L'objet du chapitre 7 porte sur les entrelaceurs hélicoïdaux. Ces derniers sont introduits dans leur plus grande généralité. Nous nous concentrons ensuite sur les tous nouveaux entrelaceurs bloc-hélicoïdaux dont il existe trois types. Nous effectuons l'étude des performances de ces entrelaceurs au sein des Codes Turbo.

Finalement, nous concluons à l'aide du chapitre 8. Mais avant de conclure sur tout le travail que nous avons effectué, nous comparons tous les entrelaceurs que nous avons présentés dans ce mémoire de façon à en faire ressortir les meilleurs dans chacun des cas considérés.

Chapitre 2 – Concaténation parallèle

2.1 Introduction

Le domaine des télécommunications est un domaine en mouvement constant. Au tout début, on se souvient des téléphones résidentiels avec fils. Nous avons ensuite connu l'avènement des MODEMS pour les ordinateurs. Depuis une décennie, le monde des télécommunications découvre l'ère du sans fil ("wireless") et surtout des transmissions numériques. Ces dernières ont permis de connaître des échanges de données de plus en plus fiables. Elles sont à l'origine du développement de ce qu'on appelle le codage correcteur d'erreur (Error Correcting Codes). Nous avons déjà mentionné Hamming qui fut un des tous premiers à introduire ces codes, mais il y a aussi les codes de Reed Solomon, et bien d'autres noms que nous pourrions citer comme Viterbi et son fameux algorithme de décodage.

En 1993 ont été introduits les Codes Turbo. Mais avant de parler de ces codes très puissants, il nous faut faire une introduction générale sur les systèmes de communications et le codage correcteur d'erreur en particulier. C'est pourquoi, nous allons, dans un premier temps, décrire les systèmes de communication. Nous nous intéresserons par la suite au codage correcteur d'erreur et en particulier aux codes convolutionnels qui sont la base des Codes Turbo. Nous verrons également certaines notions nécessaires à la compréhension de la suite du mémoire comme différents types de canaux de transmission et modulation. Enfin, nous introduirons les Codes Turbo.

2.2 Systèmes de communication

Un système de communication est un moyen de transport de l'information d'un usager à un autre, le plus souvent éloigné. Le système que nous considérons est numérique : une

séquence de symboles d'un alphabet est utilisée pour transmettre cette information. Dans notre cas, il s'agit de l'alphabet binaire. Comme nous l'avons déjà mentionné, utiliser les transmissions numériques permet en particulier de pouvoir corriger les erreurs. Les différents éléments d'un système de communication sont représentés à la figure 2.1. Différentes étapes y sont illustrées, comme le codage de source, le cryptage et le codage de canal. Nous allons présenter ces différentes étapes.

La première opération qui est effectuée sur la source de données est le codage de source [50]. Ce dernier est principalement utilisé pour réduire la redondance dans la séquence d'information. Il existe deux types de sources de données. Tout d'abord, la source de données la plus répandue est analogique. Il s'agit tout simplement de signaux électriques. Par exemple, pour un cellulaire, la source de données sera le microphone de l'appareil. En général, lorsque nous sommes confrontés à ce type de source de données, pour pouvoir effectuer du codage, il est nécessaire de numériser l'information d'origine. Ceci nous amène alors au deuxième type de données qui est tout simplement une source numérique. En effet, le numérique se retrouve en grande partie dans les ordinateurs. Lorsque nous parlons de transmissions à l'aide des ordinateurs, la source de données est directement numérique et n'a nullement besoin d'être numérisée. Dans ce cas, le codage de source s'applique directement à la source de données.

La plupart du temps, la source est analogique. Il faut donc la numériser. Le signal qui provient de la source doit, dans un premier temps, être échantillonné. Ensuite, il faut quantifier et enfin le transformer en une suite binaire de "0" et de "1". Cette transformation s'effectue à l'aide de techniques de conversion analogique-numérique comme la modulation à impulsion codée (PCM).

Après avoir codé la source, il s'agit de passer à l'étape de cryptage. Cette dernière a pour principal but d'empêcher les usagers non autorisés de comprendre le message transmis à l'aide de la source de données. Ensuite, le codage de canal est effectué. Ce dernier

consiste à ajouter de la redondance au message de façon à pouvoir corriger les erreurs qui pourraient se transmettre dans le canal. Ces erreurs sont le plus souvent introduites par du bruit qui pourrait faire passer un bit "0" à l'état de bit "1". Les trois étapes de codage que nous venons de décrire sont à effectuer dans l'ordre indiqué. Enfin, le modulateur joue le rôle d'adapteur de canal. Son rôle est de faire en sorte que la séquence convienne au canal physique par lequel les données sont transmises.

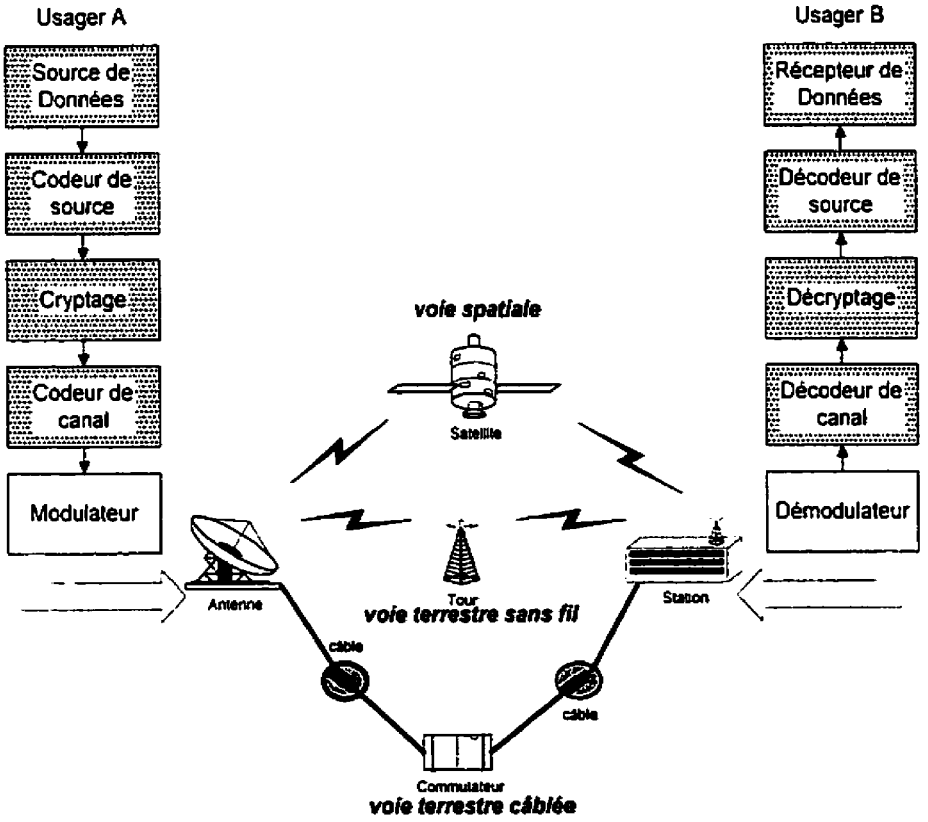


Figure 2.1 – Système de communication

La source de données ainsi modifiée passe au travers d'un canal physique. Il existe trois types de canaux différents. Le premier canal est spatial. Le signal est émis par satellite via une station terrestre. Le deuxième type de canal est le genre de canaux que nous retrouvons dans les transmissions radiomobiles comme les cellulaires. Ce type de

transmission est aussi appelé "wireless". Enfin, un troisième type de canal existe. Il s'agit des liaisons par câble, le plus souvent via de la fibre optique. On peut penser aux liaisons téléphoniques mais aussi au tout nouveau mode de transmission de l'Internet qu'offre, par exemple, la compagnie montréalaise Vidéotron. Nous nous intéresserons plus loin dans ce mémoire à ces différents types de canaux.

Ainsi, le signal est transmis vers le récepteur. Néanmoins, nous avons effectué plusieurs opérations sur ce dernier, ce qui implique évidemment qu'à la réception, nous ne recevons pas le signal d'origine. Si une voix est transmise via un cellulaire, ce que nous obtenons pour le moment n'est qu'une séquence de "1" et de "0". Pour retrouver le signal d'origine, c'est-à-dire le message vocal transmis, il faut effectuer les étapes inverses de ce que nous avons décrit ci-dessus. Évidemment, ces étapes doivent être effectuées dans l'ordre inverse. Le signal est donc tout d'abord démodulé pour ensuite être décodé par le décodeur de canal. Le décodage est en général un algorithme bien particulier correspondant au codage d'origine qui va minimiser la probabilité d'erreur. Enfin, le signal sera décrypté et le décodage de source terminera le processus.

Ayant décrit un système de communication dans son ensemble, nous allons maintenant nous intéresser au codage de l'information et tout particulièrement au codage convolutionnel.

2.3 Codage Convolutionnel

Les codes convolutionnels font partie des codes correcteurs d'erreur et sont parmi les plus utilisés dans les communications sans fil où une probabilité d'erreur par bit faible est requise. Il existe deux types de codage, à savoir le bloc et le convolutionnel. Les codeurs blocs prennent une séquence d'entrée et la séparent en plusieurs blocs de N bits. Le codeur doit donc attendre d'avoir eu N bits à son entrée pour effectuer son opération

d'encodage et transmettre son bloc codé. Les codeurs convolutionnels ont une différente approche dans le sens où le flux de bits à l'entrée et à la sortie est continu. Il ne sépare pas les bits d'entrées en blocs. Quand un bit entre dans le codeur, un ou plusieurs symboles codés sont générés à la sortie. Les codes convolutionnels ont été introduits en 1955 par Elias [19]. Ce dernier montra qu'une certaine redondance peut être introduite à l'aide de registres à décalage.

Un codeur convolutionnel binaire (n,k) est un dispositif qui accepte des k-tuples binaires en entrée et produit des n-tuples binaires en sorties. Il existe deux types de codeurs convolutionnels. Tout d'abord, un codeur convolutionnel à réponse finie à une impulsion est aussi connu sous le nom de codeur convolutionnel non récursif. Il existe aussi les codeurs convolutionnels à réponse infinie à une impulsion, qu'on appelle aussi codeur récursif. Mentionnons aussi l'appellation de systématique qui décrit un codeur dont l'information présente à l'entrée est reproduite intégralement à la sortie. Les trois principaux paramètres qui caractérisent un codeur convolutionnel sont :

- Sa longueur de contrainte K : il s'agit du nombre de bits de cases de registre nécessaires à la génération d'un bit de sortie;
- Son taux de codage R : en général, il s'agit d'une fraction dont le numérateur correspond au nombre d'entrées pour un nombre de sorties correspondant au dénominateur;
- Ses générateurs $G_1, \ldots, G_v$: ceux-ci caractérisent le code et sont les opérations binaires qui permettent d'obtenir les symboles codés de sortie.

2.3.1 Codeurs convolutionnels non récursifs

Intéressons-nous maintenant aux codeurs convolutionnels non récursifs. La figure 2.2 représente l'exemple simple d'un codeur convolutionnel de taux $R = ½$ et de longueur de contrainte $K = 3$. À l'entrée, les bits d'information d_t sont introduits un à la fois de façon

continuelle. Ils sont insérés dans un registre à décalage dont la longueur est égale à K. La mémoire du codeur ainsi défini est $\mu = K - 1$.

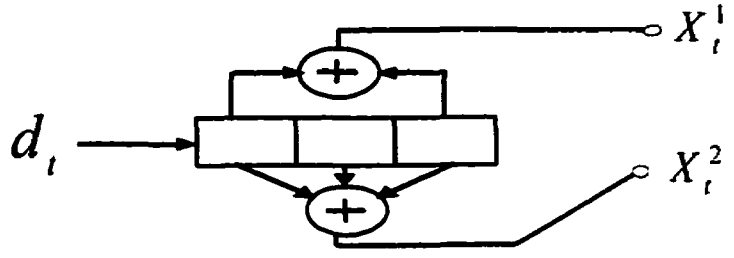


Figure 2.2 – Codeur convolutionnel non récursif et non systématique,
$G(D) = [1 + D^2 \quad 1 + D + D^2]$

À la sortie, comme nous avons un taux de codage de ½, deux symboles de parité sont générés. Ces derniers sont définis par l'addition modulo 2 des bits d'information. Les additions sont définies par les générateurs du code. Qu'entendons-nous par vecteurs générateurs? Sur la figure 2.2, nous constatons qu'il y a des additionneurs. La connexion d'un additionneur i (i = 1, 2, ..., V) à une cellule j (j = 0, 1, ..., K – 1) est définie par le paramètre binaire g_{ij}. Ce paramètre est égal à 1 s'il y a une connexion et 0 s'il n'y en a pas. Pour un i donné, nous pouvons définir un vecteur générateur que nous noterons G_i. Dans le cas de notre figure, les vecteurs générateurs sont:

$$\begin{cases} G_1 = (101) \\ G_2 = (111) \end{cases} \tag{2.1}$$

Ceci est exprimé en base binaire. Néanmoins, ces vecteurs générateurs sont le plus souvent exprimés en base octale. Ainsi, dans notre cas, nous obtiendrons :

$$\begin{cases} G_1 = 5 \\ G_2 = 7 \end{cases} \tag{2.2}$$

De façon mathématique, soit d_t le bit d'information à l'entrée du codeur à l'instant t, nous pouvons écrire que le symbole codé correspondant X_t^i peut s'écrire :

$$X_t^i = \sum_{j=0}^{K-1} g_{ij} d_{t-j} \pmod 2 \qquad i = 1, \ldots, V \tag{2.3}$$

La séquence de sortie est donc une combinaison linéaire des entrées présentes et passées. Cette séquence peut s'exprimer sous la forme du produit de convolution de la séquence d'entrée et de la réponse impulsionnelle du codeur (réponse à l'entrée 1000...), d'où le nom de codes convolutionnels.

Il est possible d'écrire cette équation sous forme de matrice, plus facile à interpréter. En introduisant la variable D comme le délai séparant deux intervalles de temps successifs, il vient que :

$$\begin{aligned}
&\overset{\rho}{X}(D) = [X_1(D), X_2(D), \Lambda\ , X_n(D)] \\
&\overset{\rho}{X}(D) = [d_1(D), d_2(D), \Lambda\ , d_k(D)] \begin{bmatrix} g_{1.1}(D) & g_{1.2}(D) & \Lambda & g_{1.n}(D) \\ g_{2.1}(D) & g_{2.2}(D) & \Lambda & g_{2.n}(D) \\ M & M & O & M \\ g_{k.1}(D) & g_{k.2}(D) & & g_{k.n}(D) \end{bmatrix} \\
&\overset{\rho}{X}(D) = \overset{\rho}{d}(D) G(D)
\end{aligned} \tag{2.4}$$

où $\quad d_u(D) = \sum_t d_t^u D^t$

La matrice G(D) est la matrice génératrice du codeur. Dans l'exemple de la figure 2.2, nous obtenons :

$$G(D) = [1 + D^2 \; 1 + D + D^2] \tag{2.5}$$

Une représentation équivalente du codeur de la figure 2.2 est illustrée à la figure 2.3. Cette représentation permet de mettre en évidence l'aspect mémoire du code. Ce genre de codeur est appelé récursif et systématique.

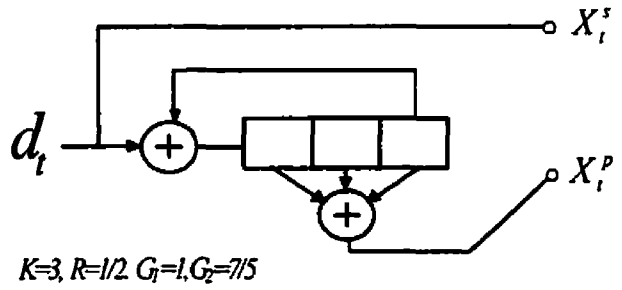


Figure 2.3 – Codeur convolutionnel récursif et systématique K = 3, R = 1/2

2.3.2 Codeurs convolutionnels récursifs

Il existe un autre type de codeur, à savoir les codeurs récursifs et systématiques. La récursivité est caractérisée par le fait que les n équations de sortie sont fonctions des entrées et sorties précédentes. Quant à l'aspect systématique, nous avons déjà mentionné qu'il s'agissait de transmettre les bits d'information dans les symboles codés. Selon Forney [21], il existe pour chaque code non systématique, un code systématique avec une boucle de contre-réaction, possédant les mêmes propriétés de distance. Pour montrer un tel processus, nous allons prendre l'exemple de la figure 2.2 et utiliser deux sorties. On a alors :

$$X^i(D) = G_i(D)d(D) \quad i = 1, 2 \tag{2.6}$$

Nous avons vu que pour avoir la propriété systématique, il faut qu'à l'une des sorties nous retrouvions exactement l'information. Nous devons donc avoir :

$$\begin{cases} \overline{X^1}(D) = d(D) = \overline{G_1}(D)d(D) \\ \overline{X^2}(D) = \dfrac{G_2(D)}{G_1(D)}d(D) = \overline{G_2}(D)d(D) \end{cases} \tag{2.7}$$

où $\overline{G} = (\overline{G_1}, \overline{G_2})$ représente le vecteur de la version systématique. Nous savons de plus que :

$$\overline{X^i}(D) = \frac{X^i(D)}{G_i(D)} \tag{2.8}$$

Nous obtenons alors les générateurs du code systématique recherché :

$$\begin{cases} \overline{G_1}(D) = 1 \\ \overline{G_2}(D) = \dfrac{G_2(D)}{G_1(D)} \end{cases} \tag{2.9}$$

Les bits de sortie peuvent alors s'écrire :

$$\begin{cases} \overline{X^1}(D) = d(D) \\ \overline{X^2}(D) = d(D)A(D) \\ A(D) = \dfrac{d(D)}{G_1(D)} \end{cases} \tag{2.10}$$

ce que nous pouvons écrire dans l'espace temporel :

$$d_t = \sum_{j=0}^{K-1} g_{1j} a_{t-j} \tag{2.11}$$

En tenant compte du fait que toutes les opérations sont faites modulo 2, et en faisant l'hypothèse que $g_{10} = 1$, le symbole a_k peut s'exprimer récursivement en fonction des symboles a_{K-j} (j = 1, 2, ..., K-1) et du symbole d_k.

$$a_t = d_t + \sum_{j=1}^{K-1} g_{1j} a_{t-j} \tag{2.12}$$

Ainsi, dans le cas d'un code récursif et systématique, ce sont les symboles a_k qui sont contenus dans le registre à décalage du codeur. À partir des équations ci-dessus, nous pouvons alors construire les symboles de parité à la sortie du codeur, l'une de ces deux séquences étant les bits d'information :

$$\begin{cases} X_t^s = d_t = \sum_{j=0}^{K-1} g_{1j} a_{t-j} \\ X_t^p = \sum_{j=0}^{K-1} g_{2j} a_{t-j} \end{cases} \tag{2.13}$$

La figure 2.4 illustre un codeur récursif systématique correspondant au codeur de la figure 2.2:

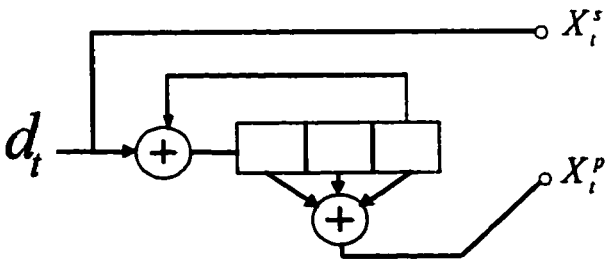


Figure 2.4 – Codeur convolutionnel récursif systématique R = ½, K = 3, G = (1,7/5)

Enfin, mentionnons les méthodes graphiques pour représenter un code convolutionnel. Il en existe trois, à savoir l'arbre, le treillis et le diagramme d'état. Ce dernier très utilisé et il est une représentation de tous les états possibles du codeur ainsi que de toutes les transitions d'un état à un autre. À la figure 2.5, nous avons illustré un exemple de ce type de diagramme. Les S_i, i = 0, 1, 2, 3 représentent les états 00, 01, 10 et 11. Les lignes en gras indiquent que le bit 1 est transmis, les lignes en pointillées que le bit 0 est transmis.

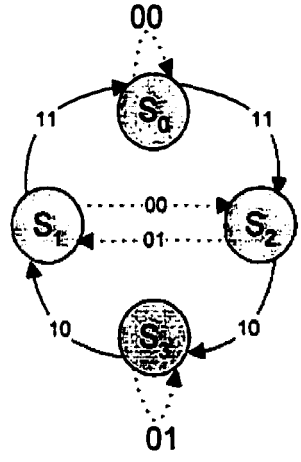


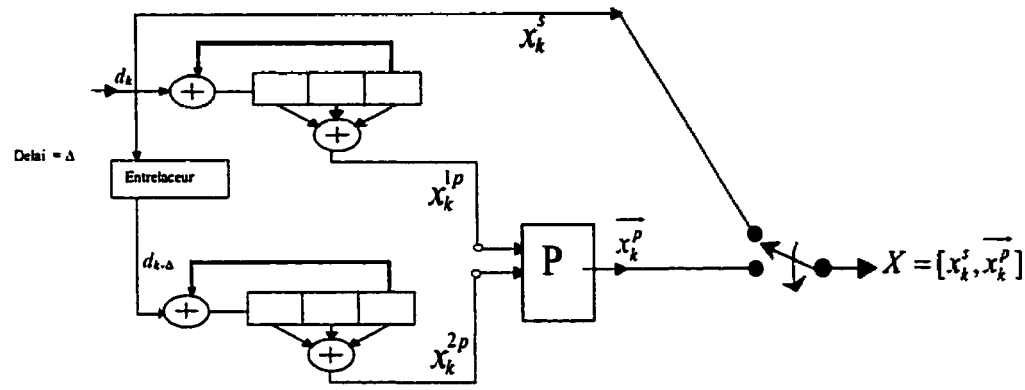
Figure 2.5 – Diagramme d'état d'un codeur convolutionnel

2.4 Concaténation parallèle

C'est à partir des codes convolutionnels que nous venons de décrire que les Codes Turbo ont été découverts. La concaténation est à la base de ces derniers. Il existe deux sortes de concaténations, à savoir la concaténation série et la parallèle. La première à avoir été introduite fut la concaténation série par Forney en 1965 [20]. En effet, ce dernier eut l'idée de relier un codeur de Reed-Solomon à un codeur convolutionnel, le tout séparé d'un entrelaceur. Le terme série indique que les deux codeurs sont mis l'un après l'autre. Cette idée de concaténation série a permis d'améliorer la probabilité d'erreur par bit de façon exponentielle. Cependant, tout gain est accompagné d'un prix que nous nous

devons de payer. Dans ce cas, le prix à payer fut la complexité. Le gain obtenu en produisant ce système s'accompagnait d'une complexité accrue dans le décodage.

Il fallut attendre trente ans pour que Berrou, Glavieux et Thitimajshima [8] décrivent ce que nous appelons maintenant la concaténation parallèle. En fait, cette dernière fut le point partant des Codes Turbo. Elle fut introduite avec la notion de décodage itératif. La différence avec la concaténation série, c'est que cette fois, les codeurs utilisés sont en parallèle. Ces derniers opèrent donc sur le même ensemble de bits plutôt que l'un sur la sortie de l'autre comme c'est la cas dans la concaténation série. Il est à noter que, que ce soit pour la concaténation série ou la parallèle, le nombre de codeurs peut être de deux ou plus. En ce qui concerne les Codes Turbo, ils ont été introduits en mettant en parallèle deux codeurs convolutionnels identiques. Par ailleurs, ces codeurs étaient récursifs et systématiques. Néanmoins, il est important de noter qu'il est possible de concevoir des Codes Turbo possédant plus de deux codeurs convolutionnels. Les Codes Turbo que nous considérons dans ce mémoire sont constitués de deux codeurs identiques de type récursifs et systématiques séparés par un entrelaceur. Il va de soit que si nous prenions trois codeurs, nous aurons deux entrelaceurs et ainsi de suite. Nous avons représenté un codeur turbo classique à la figure ci-dessous.



P : Matrice de Perforation

Figure 2.6 – Codeur turbo

Intéressons-nous maintenant au fonctionnement des Codes Turbo. Nous allons expliquer ces derniers à partir de la figure 2.6. Nous observons que ce codeur possède deux étages. Le premier étage correspond à la transmission de l'information d'entrée, c'est-à-dire que les bits sont transmis tels quel. Ce que nous appelons le deuxième étage correspond à la génération des symboles de parité permettant la correction des erreurs. Cet étage produit deux symboles de parité pour chaque symbole d'information transmis.

Lorsqu'une séquence de symboles d_k arrive au codeur, elle passe par deux étapes parallèles. La première correspond au premier codeur de l'étage supérieur. Cette étape est simplement le codage convolutionnel de cette séquence. Elle produit alors une séquence de symboles de parité x_k^{1p}. La séquence d'entrée passe en parallèle par le codeur inférieur après avoir été entrelacée. Ce deuxième codeur produit une séquence de symboles de parité x_k^{2p}. Une fois ces deux symboles de parité générés, ils peuvent être perforés ou non afin de produire la séquence ou le vecteur de parité $\overrightarrow{x_k^p} = (x_k^{1p}, x_k^{2p})$ qui sera multiplexé avec les symboles d'information $\overrightarrow{x_k^s}$. Le but de la perforation, si perforation il y a, est de supprimer certains symboles de parité afin de faire varier le taux de codage.

Si nous considérons la concaténation parallèle de deux codeurs systématiques dont les taux de codage sont $R_1 = \frac{b}{V_1}$ et $R_2 = \frac{b}{V_2}$, le taux global du codeur turbo est :

$$R = \frac{b}{V_1 + V_2 - b} = \frac{b}{\frac{b}{R_1} + \frac{b}{R_2} - b} \text{ bits/symboles} \qquad (2.14)$$

La soustraction, au dénominateur, de b est due au fait que les symboles systématiques ne sont transmis qu'une seule fois. Cette dernière équation s'écrit aussi :

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} - 1 \qquad (2.15)$$

Dans notre cas de la figure 2.6, les taux de codage R_1 et R_2 sont tous deux de ½. Le taux global de notre codeur turbo sans perforation est alors 1/3.

2.5 Modulation BPSK

Il existe de nombreux types de modulation pour les communications numériques. Néanmoins, pour notre recherche, nous n'en avons utilisé qu'une, à savoir la modulation BPSK (Binary Phase Shift Keying). Cette dernière est une modulation très utilisée dans l'étude théorique des communications numériques [37]. Effectivement, elle est caractérisée par une faible probabilité d'erreur par bit. Un modulateur BPSK est représenté à la figure ci-dessous.

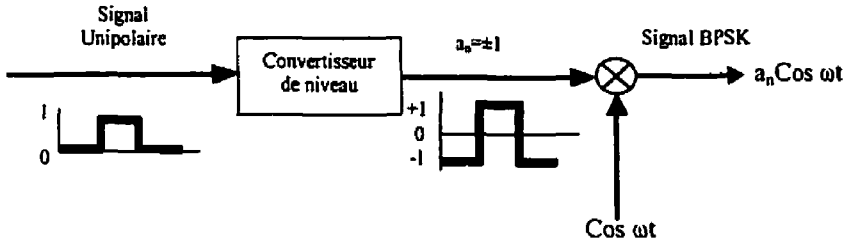


Figure 2.7 - Modulateur BPSK

En appliquant les sorties X_k et Y_k du codeur turbo à l'entrée du modulateur, nous obtenons :

$$\begin{cases} \widetilde{X}_k = 2X_k - 1 \\ \widetilde{Y}_k = 2Y_k - 1 \end{cases} \tag{2.16}$$

En BPSK, les phases opposées de la porteuse (0 et π) sont transmises toutes les T secondes en considérant T la durée temporelle d'un bit. Ces phases sont aussi représentées par +1 et −1. Ces derniers viennent des expressions $\cos(\omega_c t)$ et $\cos(\omega_c t+\pi)$. Ceci se représente sous forme de constellation à deux points. Chaque bit entrant dans le modulateur BPSK se retrouvera en l'un de ces deux points de cette constellation. Ceci est alors une modulation par phase.

Étant donné que nous avons affaire à une modulation par phase, le problème du démodulateur est de récupérer la bonne phase pour démoduler. Ceci constitue la principale difficulté du récepteur BPSK. Enfin, écrivons l'expression de la probabilité d'erreur par bit d'un tel modulateur :

$$P_b = Q(\sqrt{\frac{2.E_b}{N_0}}) \qquad \text{avec} \quad Q(x) = \int_x^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx, x > 0 \qquad (2.17)$$

2.6 Les types de canaux

Le canal est un véritable problème pour les transmissions de données. En effet, il est source de bruits de toutes sortes. Il existe des modèles pour identifier ce bruit et nous allons nous évertuer à en décrire quelques-uns.

2.6.1 Canal AWGN (Additive White Gaussian Noise)

Le premier modèle est le plus utilisé en théorie. Il s'agit du bruit additif blanc et gaussien (AWGN). Il fournit un modèle presque parfait pour certains systèmes de communication et se prête à des calculs relativement faciles. Ce modèle implique que le bruit du canal est une variable aléatoire n qui s'additionne au signal modulé. Dans ce cas, la variable n est gaussienne, de moyenne nulle et de variance σ^2. Mathématiquement, nous pouvons écrire la densité de probabilité du bruit additif blanc gaussien :

$$p(n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{n^2}{2\sigma^2}} \qquad (2.18)$$

La densité spectrale unilatérale de ce type de bruit est constante de valeur N_0. Conséquemment, si nous considérons la variable du signal modulé x_k, il nous est possible de donner la sortie du canal comme :

$$r_k = x_k + n \qquad (2.19)$$

Le modèle de bruit additif blanc gaussien est très simple et très pratique, comme nous l'avons dit, pour les calculs théoriques. Il reflète également très bien la réalité dans les communications spatiales. Néanmoins, si nous nous intéressons maintenant aux communications terrestres comme les communications radiomobiles, ce type de bruit ne reflète plus très bien la réalité. Il nous faut donc prendre un autre modèle.

2.6.2 Canal de Rayleigh

Sur la terre, les signaux sont en général soumis à une forte atténuation. Cette atténuation est le plus souvent reliée à la vitesse du récepteur, par exemple, lorsqu'on téléphone au moyen d'un cellulaire dans une voiture. Cette atténuation n'est pas modélisée par les bruit AWGN. Toutefois, il existe un modèle qui prend en compte cette atténuation. Il s'agit du modèle de canal de Rayleigh.

Le modèle de canal de Rayleigh est caractérisé par deux paramètres et est illustré à la figure 2.8. L'un de ces deux paramètres est une variable aléatoire n de distribution gaussienne. L'autre variable est l'enveloppe du signal, a_k.

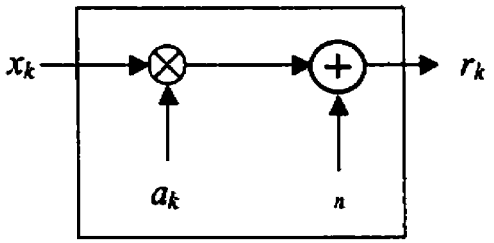


Figure 2.8 – Modèle de canal de Rayleigh

La sortie r_k du modèle de canal de Rayleigh est :

$$r_k = a_k x_k + n \tag{2.20}$$

La fonction de densité de a_k est :

$$p(a_k) = a_k e^{\frac{a_k^2}{2}}, \qquad a_k \geq 0 \tag{2.21}$$

Le modèle d'un canal de Rayleigh est schématisé à la figure 2.8, où r_k est la version bruitée du signal x_k à l'entrée du canal. La méthode la plus simple pour obtenir l'enveloppe dont la puissance moyenne est unitaire est de générer deux variables gaussiennes b_k et c_k de variance ½ et de moyenne nulle. En considérant que le processus d'évanouissement est décorrélé, l'enveloppe du signal sera alors donnée par :

$$a_k = \sqrt{b_k^2 + c_k^2} \tag{2.22}$$

2.7 Conclusion

Dans ce chapitre, nous avons tout d'abord introduit les systèmes de communication de façon générale. Nous nous sommes, par la suite, intéressés au codage convolutionnel et en particulier aux codeurs récursifs qui sont utilisés pour les Codes Turbo. La concaténation parallèle a ensuite été décrite, introduisant ainsi le codage turbo. Enfin, nous avons mentionné la modulation que nous avons utilisée pour notre système de codage et les deux types de canaux les plus utilisés dans les systèmes de communication.

Nous avons ainsi décrit la partie codage de notre système. Il nous reste maintenant à parler du décodage et en grande partie de l'algorithme de décodage que nous utilisons pour les Codes Turbo. Le prochain chapitre fait ainsi référence à l'algorithme MAP.

Chapitre 3 - Algorithmes de décodage

3.1 Introduction

Au chapitre précédent, nous avons décrit les bases de notre système de Code Turbo. Dans ce genre de système, il existe une partie codage et une partie décodage. Dans cette deuxième partie, l'élément le plus important est l'algorithme de décodage car c'est ce dernier qui permet en grande partie de minimiser la probabilité d'erreur par bits, ce que nous recherchons.

De nombreuses recherches ont été effectuées dans ce domaine. Un premier algorithme très souvent utilisé pour les Codes Turbo est le SOVA (Soft Output Viterbi Algorithm) [4]. Ce dernier est une variante de l'algorithme de Viterbi. En 1966, Chang et Hancock [14] ont développé un autre algorithme minimisant la probabilité d'erreur par symbole. Cet algorithme est appelé MAP pour Maximum a Posteriori. En 1972, Bahl et al. [1] ainsi que McAdam et al. [3] ont adapté cet algorithme aux codes correcteurs d'erreur.

La principale caractéristique des deux algorithmes dont nous venons de parler est qu'ils donnent une mesure de fiabilité sous forme probabiliste, à savoir une métrique. Nous ne nous intéresserons pas au premier de ces algorithmes, mais décrirons le deuxième. En effet, l'algorithme MAP est celui que nous avons utilisé dans notre système. Dans un premier temps, nous allons introduire la notion de "maximum a posteriori probability". Ensuite, nous nous intéresserons à l'algorithme MAP au sein des codes convolutionnels. Enfin, nous décrirons différentes versions de cet algorithme.

3.2 Probabilité a posteriori

L'algorithme de Viterbi est un algorithme très utilisé dans les systèmes de communications. Le but de celui-ci est de minimiser la probabilité d'erreur d'un mot ou d'une séquence de symboles. Dans ce cas, la règle de décision est le principe de maximum de vraisemblance. À la différence de l'algorithme de Viterbi, l'algorithme MAP agit sur la probabilité d'erreur par symbole ou par bit. La règle utilisée dans le cas de ce dernier est la maximisation de la probabilité à posteriori.

Nous allons maintenant expliquer le terme de maximum a posteriori. Dans notre cas, celui de l'algorithme MAP, nous parlerons de décision pondérée. En effet, les symboles sont décodés à l'aide d'une information supplémentaire. Pour définir la probabilité a posteriori, soient les événements A_i, $i = 1, 2, ..., M$. Ceci constitue l'ensemble des messages transmis dans un intervalle de temps donné. Maintenant, considérons B, le signal reçu que nous désirons décoder. Ce message a été corrompu par du bruit. Il s'agit d'une variable aléatoire dont nous noterons la densité de probabilité p(B). La probabilité a posteriori est définie par la probabilité que nous reconnaissions le message A_i étant donné que nous avons reçu le message B. Utilisant la règle de Bayes, on peut écrire :

$$P(A_i|B) = \frac{p(B|A_i)P(A_i)}{p(B)} \tag{3.1}$$

Essayons maintenant de définir cette probabilité plus adéquatement. En notons R la séquence reçue et d_k le symbole reçu, cette équation devient :

$$P(d_k = i|R) = \frac{P(R|d_k = i)P(d_k = i)}{P(R)} \qquad i = 0,1 \tag{3.2}$$

3.3 Maximum a posteriori

L'équation (3.2) développée au paragraphe précédent exprime la probabilité a posteriori. Nous avons déjà mentionné que le principe de l'algorithme MAP consistait à choisir le maximum de cette probabilité. Nous voyons que l'équation est différente selon que i = 0 ou i = 1. La règle de décision de l'algorithme MAP consiste alors à comparer les probabilités pour ces deux cas et de prendre le maximum, c'est-à-dire la décision la plus probable. Deux cas peuvent se présenter :

$$P(d_k = 1|R) \underset{H_1}{\overset{H_0}{\lessgtr}} P(d_k = 0|R) \tag{3.3}$$

Nous le voyons, deux hypothèses sont à considérer, à savoir l'hypothèse H_0 qui implique que le bit "0" a été transmis et l'hypothèse H_1 qui sous-entend le contraire. Insérons maintenant l'équation (3.2) dans l'équation (3.3). Il vient que :

$$\frac{p(R|d_k = 1)P(d_k = 1)}{p(R)} \underset{H_1}{\overset{H_0}{\lessgtr}} \frac{p(R|d_k = 0)P(d_k = 0)}{p(R)} \tag{3.4}$$

Nous pouvons alors simplifier cette double inéquation :

$$p(R|d_k = 1)P(d_k = 1) \underset{H_1}{\overset{H_0}{\lessgtr}} p(R|d_k = 0)P(d_k = 0) \tag{3.5}$$

que nous pouvons mettre aussi sous la forme plus conventionnelle :

$$\frac{p(R|d_k=1)P(d_k=1)}{p(R|d_k=0)P(d_k=0)} \underset{H_1}{\overset{H_0}{\lessgtr}} 1 \tag{3.6}$$

En général, la probabilité que l'on ait à priori un bit "1" ou un bit "0" à l'entrée est équiprobable. Il est donc justifié de dire que :

$$P(d_k=0) = P(d_k=1) \tag{3.7}$$

ce qui nous simplifie de façon intéressante l'équation (3.6)

$$\frac{p(R|d_k=1)}{p(R|d_k=0)} \underset{H_1}{\overset{H_0}{\lessgtr}} 1 \tag{3.8}$$

Or, grâce à la règle de Bayes, on peut écrire :

$$\begin{cases} p(R|d_k=1) = \dfrac{p(d_k=1|R)P(R)}{P(d_k=1)} \\ p(R|d_k=0) = \dfrac{p(d_k=0|R)P(R)}{P(d_k=0)} \end{cases} \tag{3.9}$$

Sachant que les bits "0" et "1" sont équiprobables, en divisant ces deux termes, (3.8) devient :

$$\frac{p(d_k=1|R)}{p(d_k=0|R)} \underset{H_1}{\overset{H_0}{\lessgtr}} 1 \tag{3.10}$$

Ceci constitue ce que nous appelons le rapport de vraisemblance. Le fait que ce soit supérieur ou inférieur à "1" implique qu'il serait judicieux de prendre le logarithme de cette expression et de comparer à zéro. Ainsi avons-nous la règle de décision suivante :

$$LLR(d_k) = \log\left[\frac{p(d_k = 1|R)}{p(d_k = 0|R)}\right] \begin{matrix} H_0 \\ < \\ > \\ H_1 \end{matrix} 0 \qquad (3.11)$$

Ceci veut donc dire que:

$$\begin{cases} \text{si } LLR(d_k) \geq 0, \text{ le bit décodé est "1"} \\ \text{si } LLR(d_k) < 0, \text{ le bit décodé est "0"} \end{cases} \qquad (3.12)$$

Ce rapport de vraisemblance est la base de l'algorithme MAP que nous allons développer. Cette règle de décision va être utilisée par la suite pour définir les étapes de cet algorithme.

3.4 Probabilité conjointe

Nous allons maintenant appliquer ce que nous avons déjà introduit au cas des Codes Turbo. Pour ce faire, nous allons considérer un codeur systématique récursif de taux de codage ½. Soit :

- $d_k = i$, $i = 0, 1$, le bit d'information présent à l'entrée du codeur à l'instant k;
- $M = K - 1$, la mémoire du codeur;
- $S_k = m$, $m = 0, 1, \ldots, 2^M$, l'état du codeur;
- N, la longueur de la séquence à coder;

- $(d_1,\Lambda,d_k,\Lambda,d_N)=(X_1^s,\Lambda,X_k^s,\Lambda,X_N^s)$, la séquence à coder (aussi la séquence d'information);
- $(X_1^p,\Lambda,X_k^p,\Lambda,X_N^p)$, la séquence composée des symboles de parité;
- $R_k=(r_k^s,r_k^p)$, la version reçue de (X_k^s,X_k^p) à l'instant k;
- $R_1^N=(R_1,\Lambda,R_k,\Lambda,R_N)$, la séquence bruitée reçue de longueur N.

Avec ces notations, l'équation du rapport de vraisemblance devient :

$$LLR(d_k)=\log\left[\frac{p(d_k=1|R_1^N)}{p(d_k=0|R_1^N)}\right] \tag{3.13}$$

Nous allons maintenant introduire une nouvelle notion. La probabilité conjointe est définie par :

$$\lambda_k^i(m)=P(d_k=i,S_k=m|R_1^N) \tag{3.14}$$

Étant donné que nous avons 2^M états :

$$LLR(d_k)=\log\left[\frac{\sum_{m=0}^{2^M-1}\lambda_k^1(m)}{\sum_{m=0}^{2^M-1}\lambda_k^0(m)}\right] \tag{3.15}$$

Maintenant que nous avons introduit la notion de probabilité conjointe, nous pouvons décrire l'algorithme MAP, ce que nous allons faire dans la prochaine section.

3.5 L'algorithme MAP

Dans cette partie, nous allons introduire de nouvelles notions qui vont nous permettre de mieux calculer le LLR. Nous allons introduire les notions de métrique d'état en avant, de métrique d'état en arrière, et de métrique de branche. En tout premier lieu, nous allons donner quelques définitions nécessaires au développement de l'algorithme MAP.

3.5.1 Définitions

Nous définissons ici trois notions essentielles au développement de la probabilité conjointe et de ce fait, à la construction de l'algorithme MAP. Ces trois notions sont :

- la métrique d'état en avant : $\alpha_k(m) = P(R_1^{k-1} | d_k = i, S_k = m, R_k^N)$ (3.16)

  $\alpha_k(m) = P(R_1^{k-1} | S_k = m)$ (3.17),

  car les événements après l'instant k n'influencent en rien les événements avant k;
- la métrique d'état en arrière : $\beta_k(m) = P(R_k^N | S_k = m)$ (3.18)
- la métrique de branche : $\delta_k^i(m) = P(d_k = i, S_k = m, R_1^k)$ (3.19)

Ayant défini ces trois notions, il s'agit maintenant de s'intéresser à la probabilité conjointe et de l'exprimer en fonction des métriques.

3.5.2 Probabilité conjointe

La probabilité conjointe est (équation (3.14)):

$$\lambda_k^i(m) = P(d_k = i, S_k = m | R_1^N) \tag{3.20}$$

Grâce à la règle de Bayes, cette expression devient, en faisant sortir R_1^N :

$$\lambda_k^i(m) = \frac{P(d_k = i, S_k = m, R_1^{k-1}, R_k^N)}{P(R_1^N)} \tag{3.21}$$

De plus, nous savons, de règle générale que :

$$P(A|B) = \frac{P(B,A)}{P(B)} \tag{3.22}$$

d'où :

$$\begin{aligned} P(d_k = i, S_k = m, R_1^{k-1}, R_k^N) = P(R_1^{k-1}|d_k = i, S_k = m, R_k^N) \\ \times P(d_k = i, S_k = m, R_k^N) \end{aligned} \tag{3.23}$$

On utilise maintenant (3.23) dans (3.21) :

$$\lambda_k^i(m) = P(R_1^{k-1}|d_k = i, S_k = m, R_k^N)\frac{P(d_k = i, S_k = m, R_k^N)}{P(R_1^N)} \tag{3.24}$$

D'où :

$$\lambda_k^i(m) = P(R_1^{k-1}|d_k = i, S_k = m, R_k^N)\frac{P(d_k = i, S_k = m, R_1^k, R_{k+1}^N)}{P(R_1^N)} \tag{3.25}$$

où on a décomposé la séquence R_1^N en $\left(R_1^k, R_{k+1}^N\right)$.

Nous allons maintenant utiliser de nouveau l'équation (3.23) de la façon suivante :

$$P(d_k = i, S_k = m, R_1^k, R_{k+1}^N) = P(R_{k+1}^N | d_k = i, S_k = m, R_1^k) P(d_k = i, S_k = m, R_1^k)$$

D'où :

$$\lambda_k^i(m) = P(R_1^{k-1} | d_k = i, S_k = m, R_k^N) P(R_{k+1}^N | d_k = i, S_k = m, R_1^k) \times \frac{P(d_k = i, S_k = m, R_1^k)}{P(R_1^N)} \quad (3.26)$$

Grâce aux métriques introduites au paragraphe précédent, nous pouvons écrire cette équation sous la forme :

$$\lambda_k^i(m) = \frac{\alpha_k(m) P(R_{k+1}^N | d_k = i, S_k = m, R_1^k) \delta_k^i(m)}{P(R_1^N)} \quad (3.27)$$

Nous n'avons pas encore introduit la métrique d'état en arrière, mais nous pouvons le faire car :

$$P(R_{k+1}^N | d_k = i, S_k = m, R_1^k) = P(R_{k+1}^N | S_{k+1} = f(i,m))$$
$$P(R_{k+1}^N | d_k = i, S_k = m, R_1^k) = \beta_{k+1}(f(i,m)) \quad (3.28)$$

En effet, en connaissant l'état du codeur $S_k = m$ et le bit d'entrée $d_k = i$ à l'instant k, l'état suivant S_{k+1} peut se noter sous la forme $S_{k+1}=f(i,m)$. Ceci signifie que l'état précédent de S_{k+1} est m et que le bit d'entrée à l'instant k est $d_k = i$.

Donc :

$$\lambda_k^i(m) = \frac{\alpha_k(m) \beta_{k+1}(f(i,m)) \delta_k^i(m)}{P(R_1^N)} \quad (3.29)$$

Finalement, avec cette expression de la probabilité conjointe, nous pouvons écrire le logarithme du rapport de vraisemblance comme suit (à l'aide de l'équation (3.15)):

$$LLR(d_k) = \log\left[\frac{\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(1,m))\delta_k^1(m)}{\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(0,m))\delta_k^0(m)}\right] \tag{3.30}$$

Nous constatons donc que le logarithme du rapport de vraisemblance est une fonction des différentes métriques définies auparavant. Le calcul de ce logarithme se réduit donc aux calculs de ces trois métriques. Nous allons maintenant nous intéresser à ces trois dernières et donner une méthode de calcul afin d'évaluer le logarithme du rapport de vraisemblance.

3.5.3 Métrique d'état en avant

Nous avons défini la métrique d'état en avant comme étant :

$$\alpha_k(m) = P(R_1^{k-1}|S_k = m) \tag{3.31}$$

Comme nous pouvons le constater, cette métrique, à l'instant k, dépend de la séquence reçue R_1^k. Cette dernière va jusqu'à l'instant k. Utilisons la règle de Bayes pour simplifier l'expression de cette métrique :

$$\alpha_k(m) = \sum_{i=0}^{2^M-1}\sum_{j=0}^{1} P(d_{k-1} = j, S_{k-1} = i, R_1^{k-1}|S_k = m) \tag{3.32}$$

$$\alpha_k(m) = \sum_{i=0}^{2^M-1}\sum_{j=0}^{1} P(R_1^{k-2}|S_k = m, d_{k-1} = j, S_{k-1} = i, R_{k-1}) \times P(d_{k-1} = j, S_{k-1} = i, R_{k-1}|S_k = m) \quad (3.33)$$

Nous savons, de plus, que l'état S_{k-1} a été initialisé par le bit d_{k-1}. Nous pouvons donc écrire S_{k-1} sous la forme $S_{k-1} = b(j,m)$. Cette notation signifie que l'état S_{k-1} est celui qui précède l'état $S_k = m$, sachant que le bit d'entrée au codeur à l'instant (k-1) est j. La métrique peut alors s'écrire :

$$\alpha_k(m) = \sum_{j=0}^{1} P(R_1^{k-2}|S_{k-1} = b(j,m)).P(d_{k-1} = j, S_{k-1} = b(j,m), R_{k-1}) \quad (3.34)$$

Or :

$$\alpha_{k-1}(m) = P(R_1^{k-1}|S_{k-1} = m) \quad (3.35)$$

Soit :

$$\alpha_{k-1}(b(j,m)) = P(R_1^{k-1}|S_{k-1} = b(j,m)) \quad (3.36)$$

De même :

$$\delta_k^i(m) = P(d_k = i, S_k = m, R_1^k) \quad (3.37)$$

Donc

$$\delta_k^i(b(j,m)) = P(d_k = i, S_k = b(j,m), R_1^k) \quad (3.38)$$

Nous reconnaissons ces termes dans l'équation (3.34). Nous en déduisons que :

$$\alpha_k(m) = \sum_{j=0}^{1} \alpha_{k-1}(b(j,m))\delta^{j}_{k-1}(b(j,m)) \tag{3.39}$$

À partir de cette dernière équation, nous constatons que cette métrique d'état à l'instant k s'exprime en fonction de la même métrique d'état à l'instant précédent. Ceci explique pourquoi nous lui donnons le nom de métrique en avant : il faut connaître les instants précédents, donc la valeur initiale pour pouvoir la calculer.

3.5.4 Métrique d'état en arrière

Au paragraphe précédent, nous avons développé l'expression de la métrique en avant. Intéressons-nous maintenant à la deuxième métrique, à savoir la métrique d'état en arrière.

Nous avons également conclu que la métrique d'état en avant dépendait des instants précédents. Il en est différemment de la métrique d'état en arrière comme l'indique son nom. En fait, cette métrique, à l'instant k, dépend de la séquence reçue (instant N) après l'instant k. Nous avons défini cette métrique comme :

$$\beta_k(m) = P(R_k^N | S_k = m) \tag{3.40}$$

Développons cette expression à l'aide de la règle de Bayes.

$$\beta_k(m) = \sum_{i=0}^{2^M-1} \sum_{j=0}^{1} P(d_k = j, S_{k+1} = i, R_k^N | S_k = m) \tag{3.41}$$

$$\beta_k(m) = \sum_{i=0}^{2^M-1} \sum_{j=0}^{1} P(R_{k+1}^N | S_k = m, d_k = j, S_{k+1} = i, R_k).P(d_k = j, S_{k+1} = i, R_k | S_k = m)$$

(3.42)

Comme pour la métrique d'état en avant, nous pouvons définir une fonction $S_{k+1} = f(j,m)$ de façon à simplifier cette expression. Il vient alors que :

$$\beta_k(m) = \sum_{j=0}^{1} P(R_k^N | S_{k+1} = f(j,m)).P(d_k = j, S_k = m, R_k) \qquad (3.43)$$

$$\beta_k(m) = \sum_{j=0}^{1} \delta_k^j(m) \beta_{k+1}(f(j,m)) \qquad (3.44)$$

Nous en déduisons que cette métrique à l'instant k est fonction de la même métrique à l'instant k+1, d'où la nomination de métrique en arrière, étant donné qu'il faut connaître l'instant k = N pour reculer dans le treillis et calculer notre métrique (elle dépend des conditions terminales).

3.5.5 Métrique de branche

Nous avons défini trois métriques et étudié deux d'entre elles. Il nous reste donc à nous interroger sur la dernière de ces métriques, à savoir celle de branche. Contrairement aux deux autres métriques, celle-ci ne dépend ni des instants précédents, ni des instants suivants. Seul l'instant présent nous intéresse. Nous avons vu que :

$$\delta_k^i(m) = P(d_k = i, S_k = m, R_1^k) \qquad (3.45)$$

Donc, toujours grâce à la règle de Bayes:

$$\delta_k^i(m) = P(R_k|d_k = i, S_k = m).P(d_k = i, S_k = m) \quad (3.46)$$

$$\delta_k^i(m) = P(R_k|d_k = i, S_k = m).P(S_k = m|d_k = i).P(d_k = i) \quad (3.47)$$

Nous savons également que $R_k = (r_k^s, r_k^s)$, donc :

$$\begin{aligned}\delta_k^i(m) = {} & P(r_k^s|d_k = i, S_k = m).P(r_k^p|d_k = i, S_k = m) \\ & \times P(S_k = m|d_k = i).P(d_k = i)\end{aligned} \quad (3.48)$$

Ceci constitue notre nouvelle expression pour cette métrique. Nous constatons qu'elle dépend des symboles recueillis à la sortie du canal. Ainsi, le type de canal va influer sur notre algorithme. Dans la suite de notre mémoire, nous allons nous intéresser au canal AWGN. Cette métrique ne changera donc pas en fonction du canal. Néanmoins, nous aurions aussi pu faire des simulations pour des canaux à évanouissement, comme le canal de Rayleigh. Dans ce cas, cette métrique aurait changé l'algorithme.

3.6 L'algorithme MAP dans un canal AWGN

Nous avons déjà défini ce type de canal. En outre, nous avons, pour ce type de canal :

- la densité de probabilité de l'information reçue est :

$$P(r_k^s|d_k = i, S_k = m) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(r_k^s - (2i-1))^2} \quad (3.49)$$

- la densité de probabilité de l'information de parité reçue est :

$$P(r_k^p|d_k = i, S_k = m) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(r_k^p - (2c_m^i - 1))^2} \quad (3.50)$$

Nous avons ici défini c_m^i le symbole codé sachant que $d_k = i$ et que $S_k = m$. Nous allons utiliser ces deux expressions pour exprimer la métrique de branche. Nous avons déjà vu que cette métrique est :

$$\begin{aligned}\delta_k^i(m) = P(r_k^s | d_k = i, S_k = m).P(r_k^p | d_k = i, S_k = m) \\ \times P(S_k = m | d_k = i).P(d_k = i)\end{aligned} \tag{3.51}$$

Nous pouvons alors remplacer les expressions des densités de probabilité définies ci-dessus dans cette dernière équation. Nous savons également qu'il y a 2^M états possibles pour le codeur. Donc le troisième terme de la dernière expression est $1/2^M$. Il vient alors que :

$$\begin{aligned}\delta_k^i(m) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(r_k^s-(2i-1))^2} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(r_k^p-(2c_m^i-1))^2} \\ \times \frac{1}{2^M} P(d_k = i)\end{aligned} \tag{3.52}$$

Nous pouvons rassembler les exponentielles :

$$\delta_k^i(m) = \frac{1}{2^M 2\pi\sigma^2} P(d_k = i) e^{-\frac{1}{2\sigma^2}\left[(r_k^s-(2i-1))^2+(r_k^p-(2c_m^i-1))^2\right]} \tag{3.53}$$

Il s'agit alors de développer cette dernière expression :

$$\begin{aligned}\delta_k^i(m) = \frac{1}{2^{M+1}\pi\sigma^2} P(d_k = i) \\ \times e^{-\frac{1}{2\sigma^2}\left[(r_k^s)^2-2r_k^s(2i-1)+(2i-1)^2+(r_k^p)^2-2r_k^p(2c_m^i-1)+(2c_m^i-1)^2\right]}\end{aligned} \tag{3.54}$$

Or, nous savons que

$$\begin{cases}(2i-1)^2=1\\(2c_m^i-1)^2=1\end{cases} \tag{3.55}$$

car les valeurs de i et c_m^i sont soit 0, soit 1.

D'où :

$$\delta_k^i(m)=\frac{1}{2^{M+1}2\pi\sigma^2}P(d_k=i)e^{-\frac{1}{2\sigma^2}\left[(r_k^s)^2-4r_k^s i+2r_k^s+1+(r_k^p)^2-4r_k^p c_m^i+2r_k^p+1\right]}$$

$$\delta_k^i(m)=\frac{1}{2^{M+1}2\pi\sigma^2}P(d_k=i) \tag{3.56}$$

$$\times e^{-\frac{1}{2\sigma^2}\left[(r_k^s)^2+(r_k^p)^2+2(r_k^p+r_k^s)+2\right]}e^{\frac{2}{\sigma^2}\left[r_k^s i+r_k^p c_m^i\right]}$$

Étant donné que nous allons effectuer le logarithme du rapport de vraisemblance, toute constante peut être enlevée. Nous pouvons alors mettre l'équation précédente sous la forme d'une constante multipliée par une fonction de la variable i et de l'état m :

$$\delta_k^i(m)=C(M,\sigma(r_k^s,r_k^p)P(d_k=i)e^{\frac{2}{\sigma^2}\left[r_k^s i+r_k^p c_m^i\right]} \tag{3.57}$$

3.7 Algorithme Log-MAP utilisé

Intéressons-nous maintenant à l'algorithme que nous allons utiliser. Nous avons vu que le logarithme du rapport de vraisemblance était :

$$LLR(d_k) = \log\left[\frac{\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(1,m))\delta_k^1(m)}{\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(0,m))\delta_k^0(m)}\right]$$

$$LLR(d_k) = \log\left[\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(1,m))\delta_k^1(m)\right] - \log\left[\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(0,m))\delta_k^0(m)\right] \quad (3.58)$$

Nous allons maintenant introduire trois nouvelles variables, à savoir les logarithmes des métriques :

$$\begin{cases} A_k(m) = -\log[\alpha_k(m)] \\ B_k(m) = -\log[\beta_k(m)] \\ D_k^i(m) = -\log[\delta_k^i(m)] \end{cases} \quad (3.59)$$

Avec ces nouvelles notations, nous obtenons :

$$LLR(d_k) = \log\left[\sum_{m=0}^{2^M-1} e^{-A_k(m)} e^{-B_k(f(1,m))} e^{-D_k^1(m)}\right] - \log\left[\sum_{m=0}^{2^M-1} e^{-A_k(m)} e^{-B_k(f(0,m))} e^{-D_k^0(m)}\right]$$

$$LLR(d_k) = \log\left[\sum_{m=0}^{2^M-1} e^{-(A_k(m)+B_k(f(1,m))+D_k^1(m))}\right] - \log\left[\sum_{m=0}^{2^M-1} e^{-(A_k(m)+B_k(f(0,m))+D_k^0(m))}\right] \quad (3.60)$$

Nous allons maintenant introduire l'opérande E entre deux variables a_1 et a_2 définie par [22][45] comme :

$$a_1 \, E \, a_2 = -\log(e^{-a_1} + e^{-a_2}) = \min(a_1, a_2) - \log(1 + e^{-|a_1 - a_2|}) \tag{3.61}$$

Soit encore, en généralisant :

$$\underset{j=0}{\overset{l}{E}} a_j = a_0 \, E \, a_1 \, E \Lambda \, E \, a_{l-1} \, E \, a_l = -\log\left[\sum_{j=0}^{l} e^{a_j}\right] \tag{3.62}$$

Avec ces notations, nous pouvons maintenant réécrire le logarithme du rapport de vraisemblance :

$$\begin{aligned} LLR(d_k) = & \underset{m=0}{\overset{2^{M-1}}{E}}\left[-(A_k(m) + B_k(f(1,m)) + D_k^1(m))\right] \\ & - \underset{m=0}{\overset{2^{M-1}}{E}}\left[-(A_k(m) + B_k(f(0,m)) + D_k^0(m))\right] \end{aligned} \tag{3.63}$$

$$\begin{aligned} LLR(d_k) = & \underset{m=0}{\overset{2^{M-1}}{E}}\left[A_k(m) + B_k(f(0,m)) + D_k^0(m)\right] \\ & - \underset{m=0}{\overset{2^{M-1}}{E}}\left[A_k(m) + B_k(f(1,m)) + D_k^1(m)\right] \end{aligned} \tag{3.64}$$

Nous avons donc une bonne expression pour le logarithme du rapport de vraisemblance. Néanmoins, il nous reste à déterminer A_k et B_k.

$$\begin{cases} A_k(m) = -\log[\alpha_k(m)] \\ \alpha_k(m) = \sum_{j=0}^{l} \alpha_{k-1}(b(j,m)).\delta_{k-1}^{j}(b(j,m)) \end{cases} \tag{3.65}$$

On peut donc en déduire le logarithme de la métrique d'état en avant :

$$
\begin{aligned}
A_k(m) &= -\log\left[\sum_{j=0}^{1}\alpha_{k-1}(b(j,m)).\delta_{k-1}^{j}(b(j,m))\right] \\
A_k(m) &= -\log\left[\sum_{j=0}^{1}e^{A_{k-1}(b(j,m))}e^{D_{k-1}^{j}(b(j,m))}\right] \\
A_k(m) &= -\log\left[\sum_{j=0}^{1}e^{A_{k-1}(b(j,m))+D_{k-1}^{j}(b(j,m))}\right] \\
A_k(m) &= \overset{1}{\underset{j=0}{E}}\left[A_{k-1}(b(j,m)) + D_{k-1}^{j}(b(j,m))\right]
\end{aligned}
\tag{3.66}
$$

Intéressons-nous maintenant à la métrique d'état en arrière :

$$
\begin{cases}
B_k(m) = -\log[\beta_k(m)] \\
\beta_k(m) = \sum_{j=0}^{1}\delta_k^{j}(m)\beta_{k+1}(f(j,m))
\end{cases}
\tag{3.67}
$$

D'où :

$$
\begin{aligned}
B_k(m) &= -\log\left[\sum_{j=0}^{1}\delta_k^{j}(m)\beta_{k+1}(f(j,m))\right] \\
B_k(m) &= -\log\left[\sum_{j=0}^{1}e^{D_k^{j}(m)+B_{k+1}(f(j,m))}\right] \\
B_k(m) &= \overset{1}{\underset{j=0}{E}}\left[D_k^{j}(m) + B_{k+1}(f(j,m))\right]
\end{aligned}
\tag{3.68}
$$

À partir des équations que nous venons de développer, il est maintenant possible de donner les différentes étapes de l'algorithme MAP :

1. **Initialisation :** à partir de l'instant k = 0,

$$\begin{cases} A_0(0) = 0 \\ A_0(m) = -\infty, \text{ pour tout } m \neq 0 \\ B_N(0) = 0 \\ B_N(m) = -\infty, \text{ pour tout } m \neq 0 \end{cases}$$

2. **Calcul et sauvegarde de la métrique de branche :** pour tous les symboles reçus, évaluer : $D_k^j(m) = \frac{2}{\sigma^2}\left(r_k^s i + r_k^p c_m^i\right)$

3. **Calcul de la métrique en avant :** à partir de l'instant k = 1 et pour tous les états : $A_k(m) = \overset{1}{\underset{j=0}{E}}\left[A_{k-1}(b(j,m)) + D_{k-1}^j(b(j,m))\right]$

4. **Calcul de la métrique en arrière :** à partir de l'instant k = N – 1 et pour tous les états : $B_k(m) = \overset{1}{\underset{j=0}{E}}\left[D_k^j(m) + B_{k+1}(f(j,m))\right]$

5. **Évaluation du rapport de vraisemblance :** pour tous les états, évaluer le LLR.

3.8 Autres versions de l'algorithme MAP

3.8.1 Algorithme log-MAP

La probabilité conjointe peut se développer de différentes façons. Ceci implique donc que les métriques qui sont concernées vont être modifiées. Une autre notation peut être adoptée pour les métrique en avant et en arrière :

$$\begin{cases} \alpha_k^i(m) = P(d_k = i, S_k = m, R_1^{k-1}) \\ \beta_k^i(m) = P(R_{k+1}^N | d_k = i, S_k = m) \end{cases} \quad (3.69)$$

Avec ce que nous avons défini lors de ce chapitre, nous sommes capables de développer le LLR qui est :

$$LLR(d_k) = \underset{m=0}{\overset{2^M-1}{E}}\left[A_k^0(m) + B_k^0(m)\right] - \underset{m=0}{\overset{2^M-1}{E}}\left[A_k^1(m) + B_k^1(m)\right] \qquad (3.70)$$

L'algorithme que nous avons décrit ne change alors que pour les équations en elles-mêmes.

3.8.2 L'algorithme MAP sous optimal

Le calcul de l'opérande E peut être très compliqué. C'est pourquoi il est important de chercher à simplifier notre algorithme. Une des façons de faire est de réduire le nombre de calculs nécessaires à l'opérande E. Une simplification qui peut être intéressante est comme suit :

$$a_1 \, E \, a_2 \approx \min(a_1, a_2) \qquad (3.71)$$

À partir de cette équation, nous pouvons alors donner le rapport de vraisemblance de l'algorithme :

$$LLR(d_k) = \min_m[A_k^0(m) + B_k^0(m)] - \min_m[A_k^1(m) + B_k^1(m)] \qquad (3.72)$$

Cet algorithme permet une implémentation matérielle plus facile. Néanmoins, il s'accompagne d'une complexité importante, à savoir l'implémentation de deux boucles récursives [44].

3.9 Comparaison de l'algorithme SOVA avec l'algorithme MAP

L'algorithme SOVA (Soft Output Viterbi Algorithm) est une version simplifiée de l'algorithme de Viterbi. Il permet la génération d'une information mesurant le degré de fiabilité de la décision prise lors du décodage des symboles reçus. Son principal avantage est qu'il nécessite une seule boucle récursive pour le calcul des probabilités d'erreur, contrairement à l'algorithme MAP. De surplus, il ne requiert aucun stockage des séquences reçues pour le calcul des branches comme le MAP. Son implémentation n'est donc que plus aisée, mais plus difficile que Viterbi.

Il a été démontré que les probabilités générées par l'algorithme SOVA étaient biaisées et ceci conduisait à une sous-évaluation de la probabilité d'erreur. Ceci a en particulier été constaté pour des milieux bruités. Enfin, il a également été démontré que la variation du niveau de sorties de l'algorithme MAP est beaucoup plus importante. Ceci implique donc une meilleure sensibilité pour ce dernier.

Il était important de mentionner qu'un autre algorithme que le MAP était possible pour le décodage turbo, à savoir le SOVA. Néanmoins, étant donné que l'algorithme MAP offre de meilleures caractéristiques, nous avons utilisé ce dernier.

3.10 Conclusion

Dans ce chapitre, nous avons présenté l'algorithme MAP que nous avons utilisé pour le décodage turbo de nos simulations. Ce dernier a été choisi car il permet la minimisation de la probabilité d'erreur par bit ainsi que la génération d'information de fiabilité pour chaque bit. Il est important de noter que cet algorithme est complexe et qu'il induit un certain délai dans le décodage. Nous avons également vu qu'il existait des simplifications

à cet algorithme. Enfin, il est important de savoir que d'autres algorithmes, comme le SOVA sont disponibles pour le décodage turbo, même si nous ne l'avons pas utilisé.

Le choix de l'algorithme de décodage est extrêmement important pour le décodage turbo. Néanmoins, il n'est pas le seul paramètre à prendre en compte étant donné que ce décodage est très complexe. Nous allons voir, dans le prochain chapitre un autre des paramètres importants au décodage, à savoir l'entrelaceur. Toutefois, avant de parler des entrelaceurs en général, nous nous évertuerons à décrire de façon détaillée le processus de décodage turbo.

Chapitre 4 – Décodage itératif et entrelacement

4.1 Introduction

Dans le chapitre précédent, nous nous sommes concentrés sur un algorithme de décodage propre au décodage itératif. Ce dernier est très approprié pour les Codes Turbo. Nous avons également vu le processus de codage de ces codes. Il nous reste maintenant à parler de la deuxième partie, à savoir le processus de décodage itératif. Ce dernier est ce qui donne aux Codes Turbo la propriété de "turbo". Nous verrons que la notion de plusieurs itérations peut apporter un énorme gain de codage.

Néanmoins, même si l'algorithme de décodage est extrêmement important pour les Codes Turbo, il n'en reste pas moins que les entrelaceurs jouent aussi un rôle important dans les performances des Codes Turbo. Un entrelaceur est un système qui prend une séquence de symboles d'un alphabet à l'entrée et qui produit une séquence de ces mêmes symboles dans un ordre différent. Le but de l'entrelaceur n'est donc pas de changer l'information, mais d'en modifier l'ordre.

Dans le cas des canaux à salves d'erreurs, le problème est qu'une salve d'erreur peut toucher un seul mot de code. Les codes ne sont pas capables de corriger les erreurs s'il y a plus d'un certain nombre de bits en erreurs dans un mot de code. Le but de l'entrelacement est alors de faire en sorte qu'une salve d'erreur se retrouve dans plusieurs mots de code différents. En séparant les bits, une salve d'erreur longue de 5 bits pourra éventuellement ne toucher qu'un bit de cinq mots de code différents.

Il existe deux sortes d'entrelaceurs classiques, le bloc et le convolutionnel. Le premier consiste à insérer des bits dans une matrice le long des lignes et de lire les bits de sortie colonne par colonne. L'entrelaceur pseudo-aléatoire est un dérivé de ce dernier étant

donné que des bits sont mis en mémoire de façon séquentielle et lus à la sortie de façon aléatoire. Le deuxième entrelaceur classique est le convolutionnel. Son principe est différent du bloc dans le sens où la notion de délai est induite. En effet, les bits sont insérés par lignes, mais chaque ligne possède un délai de plus que la précédente, la première ligne n'ayant aucun délai. Nous aurons l'occasion de présenter en détail ces deux types d'entrelaceur plus loin dans ce chapitre.

4.2 Principe de décodage itératif

Nous avons vu que les symboles fournis par le canal au décodeur sont de trois sortes. Il s'agit en effet des symboles d'information et des symboles de parité générés par le premier codeur et ceux par le deuxième. Le décodage itératif va donc prendre ces trois types de symboles et minimiser la probabilité d'erreur par bit. Le principe de décodage itératif est illustré à la figure 4.1 ci-dessous :

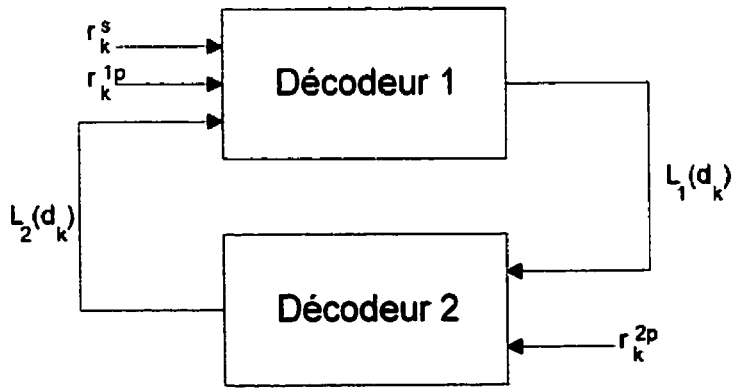


Figure 4.1 – Principe de décodage itératif

La technique de décodage illustrée par cette figure se fait par itérations de deux étapes. En effet, comme nous le voyons, contrairement au codage, les blocs (décodeurs) sont en série et non en parallèle. Il va donc de soi que chaque décodeur agit l'un après l'autre. À partir des séquences reçues $\{r_k^s\}$ et $\{r_k^{1p}\}$, le décodeur numéro 1 décode le bloc reçu. Il

transmet une information $L_1(d_k)$ qui est appelée information extrinsèque et liée au logarithme du rapport de vraisemblance. Cette information peut être vue comme une évaluation de la fiabilité du décodage de décodeur 1. Le décodeur 2 a alors à sa disposition cette information de fiabilité ainsi que les symboles de parité du deuxième codeur. Ainsi, certaines erreurs non corrigées par le décodeur 1 peuvent l'être par le décodeur 2. Ces deux étapes se répètent autant de fois que désiré par une boucle de contre réaction. Notons que nous avons omis l'entrelaceur sur ce schéma pour une explication plus claire.

Considérons maintenant l'information extrinsèque entre les deux décodeurs. Supposons que les LLR générés par le décodeur 1 soient complètement générés. $L_1(d_k)$ est alors fonction de $\{r_k^s\}$ et $\{r_k^{1p}\}$. De la même façon, la génération de l'information extrinsèque $L_2(d_k)$ du deuxième décodeur est fonction de $\{r_k^{2p}\}$ et de $L_1(d_k)$. Comme $L_1(d_k)$ est fonction des séquences $\{r_k^s\}$ et $\{r_k^{1p}\}$, il en est de même de $L_2(d_k)$ car cette dernière est fonction de $L_1(d_k)$. Ainsi, l'information extrinsèque que va recevoir décodeur 1 à la deuxième itération est fonction des séquences reçues à la première itération par ce même décodeur. Nous allons donc avoir une certaine corrélation indésirable entre le premier et le deuxième décodeur. C'est pourquoi, et nous le verrons plus loin, seule une partie (celle non corrélée) de l'information extrinsèque est transmise. Cette information extrinsèque fait l'objet de notre prochain paragraphe.

4.3 Information extrinsèque

Nous allons maintenant manipuler mathématiquement l'information extrinsèque de façon à extraire l'information nécessaire au décodage turbo. Nous avons déjà vu, au chapitre précédent que :

$$LLR(d_k) = \log\left[\frac{\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(1,m))\delta_k^1(m)}{\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(0,m))\delta_k^0(m)}\right] \quad (4.1)$$

Nous avons également vu que pour un canal AWGN, on avait :

$$\delta_k^i(m) = C(M,\sigma(r_k^s, r_k^p)P(d_k = i)e^{\frac{2}{\sigma^2}\left[r_k^s i + r_k^p c_m^i\right]} \quad (4.2)$$

Il vient donc que, en insérant (4.2) dans (4.1) :

$$LLR(d_k) = \log\left[\frac{\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(1,m))C(M,\sigma,r_k^s,r_k^p)P(d_k = 1)e^{\frac{2}{\sigma^2}\left[r_k^s + r_k^p c_m^1\right]}}{\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(0,m))C(M,\sigma,r_k^s,r_k^p)P(d_k = 0)e^{\frac{2}{\sigma^2}\left[r_k^p c_m^0\right]}}\right] \quad (4.3)$$

Il s'agit de cette information qui est transmise au décodeur 1 ou au décodeur 2. Seulement, une partie de cette information ne sera pas transmise, à cause de la corrélation qu'elle pourrait avoir avec la nouvelle séquence reçue. Manipulons donc cette équation :

$$LLR(d_k) = \log\left[\frac{P(d_k = 1)e^{\frac{2}{\sigma^2}r_k^s}\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(1,m))C(M,\sigma,r_k^s,r_k^p)e^{\frac{2}{\sigma^2}r_k^p c_m^1}}{P(d_k = 0)\sum_{m=0}^{2^M-1}\alpha_k(m)\beta_{k+1}(f(0,m))C(M,\sigma,r_k^s,r_k^p)e^{\frac{2}{\sigma^2}\left[r_k^p c_m^0\right]}}\right]$$

$$LLR(d_k) = \log\left[\frac{P(d_k = 1)}{P(d_k = 0)}\right] + \log\left[e^{\frac{2}{\sigma^2} r_k^s}\right] + \log\left[\frac{\sum_{m=0}^{2^{M-1}} \alpha_k(m)\beta_{k+1}(f(1,m))C(M,\sigma,r_k^s,r_k^p)e^{\frac{2}{\sigma^2} r_k^p c_m^1}}{\sum_{m=0}^{2^{M-1}} \alpha_k(m)\beta_{k+1}(f(0,m))C(M,\sigma,r_k^s,r_k^p)e^{\frac{2}{\sigma^2}\left[r_k^p c_m^0\right]}}\right] \quad (4.4)$$

L'équation (4.4) s'écrit alors aussi :

$$LLR(d_k) = L_a(d_k) + \frac{2}{\sigma^2} r_k^s + L(d_k) \quad (4.5)$$

Nous voyons donc que ce logarithme du rapport de vraisemblance se décompose en trois termes. Le premier terme correspond à ce qu'on appelle l'information à priori. Cette information est un indice supplémentaire sur le bit qui va être décodé. Le deuxième terme indique l'influence du canal sur ce logarithme. Quant au troisième, il s'agit de l'information extrinsèque qui est l'information qui sera utilisée à titre de quantificateur de la justesse du décodage. Cette information extrinsèque est très importante pour le décodage turbo car elle fait circuler le terme de correction d'un décodeur à un autre.

De plus, pour la première itération, il est concevable d'émettre l'hypothèse d'équiprobabilité pour les bits "0" et "1". Donc :

$$\log\left[\frac{P(d_k = 1)}{P(d_k = 0)}\right] = 0$$

$$LLR(d_k) = \frac{2}{\sigma^2} r_k^s + L(d_k) \qquad (4.6)$$

4.4 Décodeur turbo

Nous avons défini l'information extrinsèque et l'algorithme de décodage. Nous pouvons maintenant préciser plus en détail le processus de décodage des Codes Turbo. Un décodeur turbo est fourni à la figure 4.2. Notons que sur cette figure apparaît ce qu'on appelle un entrelaceur. Pour le moment, nous ne parlerons pas de ce système et allons décrire le processus.

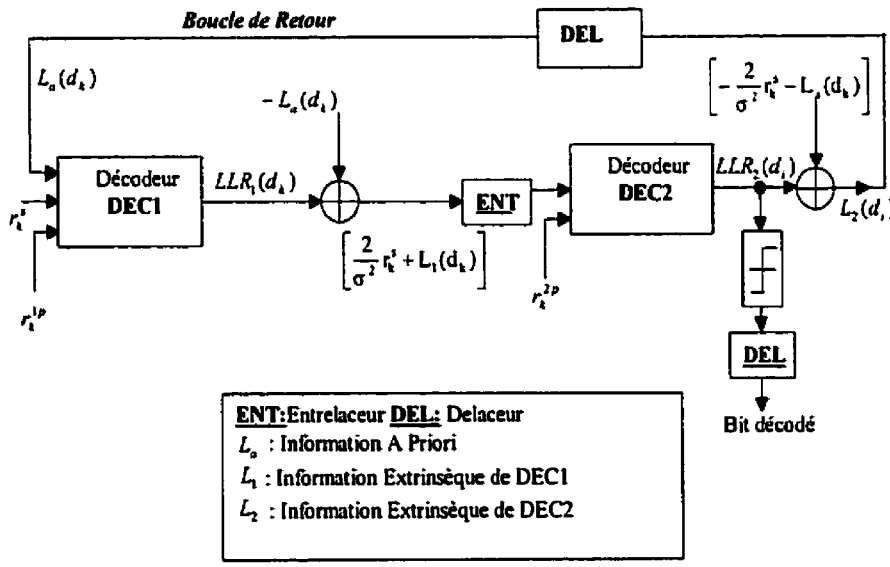


Figure 4.2 – Schéma d'un décodeur turbo

Les bits sont reçus par le premier décodeur et une certaine information est transmise au deuxième. Nous venons de voir, au paragraphe précédent, que le logarithme du rapport de vraisemblance du premier décodeur est $\frac{2}{\sigma^2} r_k^s + L(d_k)$. Pour fins d'explications, nous

allons renommer ce terme car nous avons deux décodeurs pour le décodage turbo. Nous allons donc dire que le deuxième décodeur reçoit l'information suivante :

$$LLR_1(d_k) = \frac{2}{\sigma^2} r_k^s + L_1(d_k) \tag{4.7}$$

L'information $L_1(d_k)$ devient alors l'information a priori pour le deuxième décodeur. Néanmoins, comme nous pouvons le constater sur la figure du décodeur turbo, il existe un entrelaceur. Ce dernier permet au deuxième décodeur de voir une séquence de mêmes bits que le premier, mais dans un ordre différent. Pour être rigoureux, nous allons alors changer l'indice du bit. Le deuxième décodeur reçoit donc l'information suivante :

$$LLR_1(d_i) = \frac{2}{\sigma^2} r_i^s + L_1(d_i) \tag{4.8}$$

Cette information passe au travers du deuxième décodeur. Ainsi, à la sortie de ce dernier, nous nous retrouvons avec l'information :

$$LLR_2(d_i) = \frac{2}{\sigma^2} r_i^s + L_1(d_i) + L_2(d_i) \tag{4.9}$$

$L_2(d_i)$ représente l'information extrinsèque du deuxième décodeur. Cette information, à l'aide d'une boucle de retour, est ensuite retransmise au premier décodeur, en ayant, au passage, été délacée pour que le premier décodeur voit la séquence dans l'ordre exact. Pour que ce premier décodeur soit efficace, il faut lui transmettre une bonne information a priori qui lui donne une efficacité sur le décodage du deuxième décodeur. Néanmoins, cette information a priori doit être différente de ce que le premier décodeur connaît déjà. Il nous faudra donc extraire l'information du décodeur 1 de la première itération. De façon à ce que ce décodeur fonctionne correctement, nous devons lui transmettre $L_2(d_k)$.

Il nous faut donc soustraire, à l'information transmise par le deuxième décodeur, le terme $\frac{2}{\sigma^2} r_i^s + L_1(d_i)$. Cette information est donc transmise au premier décodeur avec un certain délai.

À la deuxième itération, le premier décodeur va transmettre une autre information au deuxième décodeur. Selon le même principe que nous avons décrit, l'information transmise au deuxième décodeur sera :

$$LLR_1(d_j) = \frac{2}{\sigma^2} r_j^s + L_1(d_i) + L_2(d_i) \qquad (4.10)$$

Pour des raisons similaires, de façon à ne transmettre que l'information a priori, seule $L_1(d_j)$ est transmise. Maintenant que nous avons décrit le processus de décodage des Codes Turbo, nous allons étudier les entrelaceurs.

4.5 Introduction aux entrelaceurs [26]

Un entrelaceur est un système à une entrée et une sortie. Il prend une séquence de symboles à l'entrée et produit une séquence du même alphabet à la sortie dans un ordre complètement différent. La notation que nous allons utiliser pour un entrelaceur de période T est la suivante :

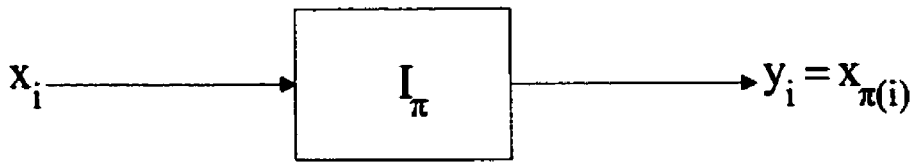


Figure 4.3 – Un entrelaceur

Le bit x_i est inséré à l'entrée de l'entrelaceur I_π à l'instant i. La sortie, après une certaine latence, est alors $y_i = x_{\pi(i)}$. Cette notation correspond au temps réel. Si maintenant, nous exprimons la même chose en terme de séquences, nous obtenons : $y = I_\pi(x)$. De façon mathématique, l'entrelaceur I_π est décrit par la permutation : $\pi : Z \rightarrow Z$. L'entrelaceur est alors une permutation sur les entiers de l'alphabet Z.

En ce qui concerne l'opération inverse, à savoir le dé-entrelaceur, il s'agit d'un système qui opère à la sortie de l'entrelaceur ou plus loin dans un système de communication et qui remet en place les symboles tels qu'ils étaient à l'origine. Nous utiliserons la notation I_μ pour le dé-entrelaceur. Avec les notations que nous utilisons, il vient tout de suite que : $I_\mu = I_\pi^{-1}$.

De façon logique, pour qu'un entrelaceur soit réalisable, il faut qu'il soit périodique de période finie. De façon mathématique, pour vérifier qu'un entrelaceur est bien de période finie T, il suffit d'effectuer une simple opération de délai sur cet entrelaceur. Si l'entrelaceur commute avec l'opération délai, alors ce dernier possède une période finie. Il est à noter que la période que nous considérons est la fondamentale, à savoir la plus petite période de l'entrelaceur. Un entrelaceur qui a une période T satisfait l'équation suivante :

$$\pi(i) - T = \pi(i\ T),\ \forall\ i \qquad (4.11)$$

Ayant maintenant introduit la notion de période finie d'un entrelaceur, il nous est possible de représenter les permutations de ce dernier. Il s'agit d'illustrer les images de chacun des éléments de l'entrelaceur. Une des façons de faire est illustrée à la figure suivante :

$$\begin{pmatrix} 0 & 1 & 2 & \Lambda & T-1 \\ \pi_0 & \pi_1 & \pi_2 & \Lambda & \pi_{T-1} \end{pmatrix}$$

Figure 4.4 – Vecteur de permutation d'un entrelaceur

Cette notation correspond a la permutation fondamentale de l'entrelaceur. Pour obtenir les autres permutations lors d'une communication numérique, il suffit de combiner cette permutation fondamentale avec la périodicité de l'entrelaceur. Notons qu'avec une telle notation, nous pouvons omettre la première ligne de ce vecteur de permutation.

Une autre façon de montrer les permutations d'un entrelaceur est illustrée à la figure 3. Néanmoins, cette notation, bien que plus graphique et permettant de mieux voir l'évolution d'un entrelaceur, est difficile à mettre en place pour un entrelaceur dont la période est grande. Cette notation reste quand même intéressante dans le cas de la théorie des entrelaceurs.

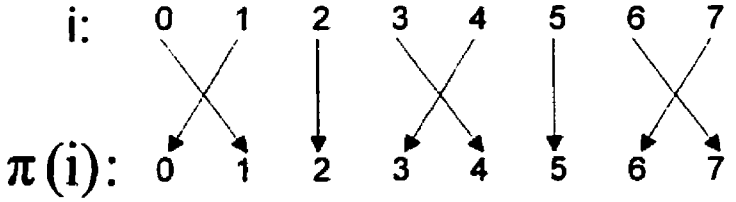


Figure 4.5 – Un entrelaceur de période 3

4.6 Méthodes d'entrelacement

Il existe deux façons de modifier l'ordre des lettres d'un alphabet, en particulier des bits. Dans un premier temps, la modification peut se faire par blocs. Ce genre d'entrelacement sera qualifié de bloc. Quant au deuxième, il s'agit de l'entrelacement multiplexé. Ce dernier type est plus connu pour son introduction de délai dans la transmission. Nous

allons décrire brièvement ces deux types d'entrelacement qui sont fondamentaux dans l'étude des entrelaceurs.

4.6.1 Entrelaceurs blocs

Il s'agit d'un entrelaceur qui effectue une permutation spécifique sur l'ensemble de départ $Z_T = \{0, 1, \Lambda, T-1\}$. Par cela, nous entendons qu'il s'agit d'une application bijective d'un ensemble sur le même ensemble. Chaque origine a une et une seule image qui fait partie du même ensemble de départ. Ainsi, l'ensemble d'arrivée est $Z_T = \{\pi_0, \pi_1, \ldots, \pi_{T-1}\}$. Ce genre d'entrelaceur est le type même qui effectue une permutation et qui possède une période fondamentale.

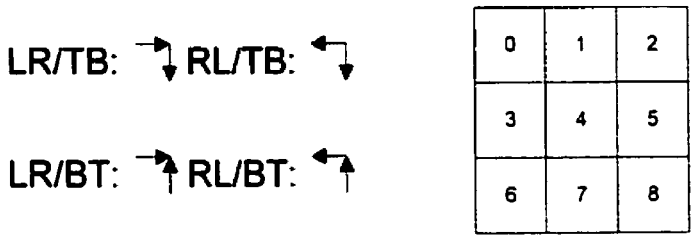


Figure 4.6 – Un exemple de processus bloc

Son nom, entrelaceur bloc, vient du fait qu'il effectue une permutation sur un bloc de symboles. Néanmoins, étant donnée de la confusion que ce nom peut apporter avec l'entrelaceur bloc classique, nous qualifierons ce genre d'entrelaceurs de permutation. En effet, le meilleur exemple pour ce genre d'entrelaceur est l'entrelaceur bloc classique. Un tel entrelaceur est illustré à la figure (4.6) avec ses quatre modes d'opération. Nous nous chargerons de décrire ce système dans la suite du mémoire.

4.6.2 Entrelaceurs multiplexés

Ce type d'entrelacement s'effectue également sur un bloc de bits, mais à la différence des entrelaceurs blocs, sa caractéristique est l'introduction de délais dans la transmission. Ce genre d'entrelacement est illustré à la figure 4.6. La séquence d'entrée est multiplexée en T sous séquences. Chaque sous séquence (correspondant à une ligne sur le schéma) introduit un délai variable. Les entrelaceurs convolutionnels font partie de ce type d'entrelaceurs.

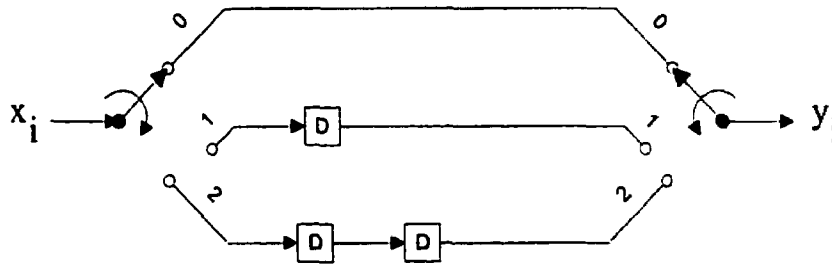


Figure 4.6 – Un entrelaceur multiplexé

4.7 Représentations et décomposition

Il s'agit maintenant de définir des paramètres nécessaires à l'étude des entrelaceurs. Ces paramètres sont utiles également au design de ces derniers. Dans un premier temps, il va s'agir d'introduire des notions qui permettront par la suite de décrire ces paramètres.

4.7.1 Décomposition

Il est possible de définir un entrelaceur à l'aide d'une matrice que nous qualifierons de génératrice. Il est important tout d'abord de noter qu'un entrelaceur, quel qu'il soit, peut être décomposé de deux façons différentes qui sont équivalentes :

- tout entrelaceur peut se décomposer en un entrelaceur de permutation suivi d'un entrelaceur multiplexé;
- tout entrelaceur peut se décomposer en un entrelaceur multiplexé suivi d'un entrelaceur de permutation.

Cette dernière notion peut être illustrée plus facilement à l'aide des matrices. Ceci va faire le sujet de notre prochain paragraphe.

Nous allons tout d'abord décrire une matrice génératrice d'un entrelaceur de permutation. Nous considérons un entrelaceur de période T. L'illustration se fait à l'aide d'une matrice carrée T x T de "1" et de "0". De façon intuitive, si un "1" apparaît dans la $i^{ème}$ ligne et la $j^{ème}$ colonne, une permutation s'effectue du $i^{ème}$ symbole de l'entrée vers le $j^{ème}$ symbole de sortie. Ce concept est illustré à la figure ci-dessous :

$$G = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad G = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Figure 4.7 – Matrices génératrices 3x3 d'entrelaceurs blocs

L'autre type d'entrelaceurs dont nous avons parlé peut également être représenté par une matrice, mais cette fois, celle-ci ne contiendra pas de 1 ni de 0. Il s'agit ici des entrelaceurs multiplexés qui introduisent un délai. Il faut donc introduire la notion de délai que nous allons noter "D". S'il n'y a aucun délai, il y aura un "1" dans la matrice. Pour un délai de 1, le symbole "D" sera utilisé, pour un délai de 2, le symbole "D^2" sera introduit et ainsi de suite. Il s'agit donc ici de la notion de monôme. Il est à noter que cette matrice est diagonale étant donné qu'un délai est introduit sur chacun des bits, il n'y a donc pas de permutation. La matrice utilisée sera également de dimensions T x T. Un exemple de ce genre de matrice est illustré à la figure 4.7:

$$G = \begin{pmatrix} 1 & 0 & 0 \\ 0 & D & 0 \\ 0 & 0 & D^2 \end{pmatrix}$$

Figure 4.7 –Matrice génératrice 3x3 d'entrelaceur multiplexé

Ayant introduit les deux types de matrices génératrices, il est maintenant facile d'illustrer la décomposition d'un entrelaceur général par un produit de matrice de permutation par une matrice multiplexée. La notion de permutation est comprise dans la première, alors que le concept de délai est introduit par la deuxième. Soit M(D) une matrice diagonale représentant la partie multiplexée de l'entrelaceur. Nous définissons ensuite P la matrice illustrant les permutations de cet entrelaceur. Alors, la matrice de l'entrelaceur est :

$$G = M_1(D).P = P.M_r(D) \tag{4.12}$$

4.7.2 Notion d'équivalence

Ayant introduit les matrices génératrices des entrelaceurs, nous allons maintenant nous intéresser à la notion d'équivalence. Nous savons qu'un entrelaceur possède en général une période sur laquelle les permutations se répètent. Fondamentalement, il est donc logique de penser que deux entrelaceurs qui possèdent les mêmes permutations, différentes dans le temps, sont équivalents. Ceci constitue la définition d'équivalence des entrelaceurs.

Nous dirons que deux entrelaceurs sont équivalents si :

$$\pi'(i) = \pi(i-n) + m \tag{4.13}$$

Cette définition est en terme de délai. Néanmoins, en terme d'application, ceci équivaut à :

$$I \circ D^n = D^m \circ I' \qquad (4.14)$$

Cela équivaut donc à dire que l'entrelaceur I est équivalent à l'entrelaceur I' si en retardant l'un et en avançant l'autre, nous obtenons les mêmes permutations. Un exemple de ceci est illustré ci-dessous :

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 3 & 2 & 1 & 0 & 7 & 6 & 5 & 4 & 11 & 10 & 9 & 8 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 4 & 3 & 2 & 1 & 8 & 7 & 6 & 5 & 12 & 11 & 10 & 9 \end{pmatrix}$$

Figure 4.8 – Un exemple de deux permutations équivalentes

4.7.3 Notion de causalité

Nous allons maintenant introduire la notion de causalité qui nous servira dans l'étude théorique des paramètres d'un entrelaceur. Un entrelaceur est dit causal si la sortie $y = x_{\pi(i)}$ au temps i dépend seulement des entrées x_j avec $j \leq i$. Cela veut donc dire que la sortie j ne dépend que des entrées précédentes et de l'entrée courante.

Les entrelaceurs en général ne sont pas causaux. Néanmoins, il est possible, à l'aide d'une opération, de les rendre causal. Pour ceci, il nous faut définir la séparation minimale entre i et π(i). Intuitivement, ceci correspond au minimum de distance géographique après entrelacement :

$$\delta = \min_{0 \leq i < T} (i - \pi(i))$$

Avec cette notation, il est facile de voir qu'un entrelaceur est causal si cette distance est positive, soit $\delta \geq 0$. Maintenant, il est possible de rendre causal tout entrelaceur à l'aide de délais appropriés sur les entrées et les sorties. En effectuant des délais de la sorte sur un entrelaceur, on obtient un entrelaceur équivalent qui possède les mêmes propriétés. Nous verrons plus loin dans ce chapitre que les paramètres d'un entrelaceur sont donnés par son entrelaceur causal équivalent.

En ce qui concerne la matrice génératrice de l'entrelaceur causal, elle possède deux propriétés :

- Les exposants de D sont tous positifs
- Tous les termes dans la partie triangulaire gauche de la matrice (les termes $g_{i,j}$ avec $j<i$) sont divisibles par D.

4.8 Paramètres d'un entrelaceur

Nous avons maintenant introduit tous les outils nécessaires à la description des paramètres des entrelaceurs. Il existe beaucoup de paramètres et nous allons, dans ce paragraphe, décrire les essentiels. Il s'agira surtout du délai, de la mémoire, et du facteur d'étalement.

4.8.1 Le délai

Le délai d'un entrelaceur est un paramètre très important dans le design. En effet, il est difficile de concevoir un délai trop important dans la transmission de données de nos jours étant donné qu'un des facteurs importants d'une transmission est la rapidité. Il est donc important de connaître à l'avance ou de pouvoir évaluer le délai que notre système va introduire.

Le délai d'un entrelaceur est le délai que la paire entrelaceur/dé-entrelaceur crée. Il s'agit du délai total des données après entrelacement et dé-entrelacement. Nous avons vu qu'il existait deux types d'entrelaceurs. Pour un entrelaceur de permutation, nous savons qu'il faut attendre que la matrice soit remplie avant d'entrelacer et de transmettre. Le délai de la transmission va donc être de l'ordre du nombre de bits dans la matrice, ce qu'on appelle la taille de l'entrelaceur. Le délai en terme de temps sera alors dépendant de la vitesse de transmission. Il en est de même pour un entrelaceur multiplexé, à la différence que ce genre d'entrelaceur nous donne directement le délai impliqué, étant donné que le schéma est représenté par des blocs de délai.

Il est toutefois très difficile d'évaluer le délai d'un entrelaceur. Ce que nous pouvons dire est que plus la longueur du bloc est grande, plus le délai sera important. Ceci est à considérer dans tout design. Maintenant, pour des communications par satellite, ce genre de paramètres est peu important. En effet, toute transmission par satellite est induite de délai.

4.8.2 Mémoire

Tout entrelaceur nécessite une certaine mémoire pour entrelacer. Nous avons déjà vu la notion de causalité. Si notre entrelaceur n'est pas causal, la mémoire requise est tout simplement la longueur de l'entrelaceur. Néanmoins, la notion de causalité permet de trouver la mémoire minimum requise pour un entrelaceur. En effet, un entrelaceur est causal si la sortie au temps i ne dépend que des entrées précédentes ou courantes. La mémoire requise est alors le nombre de bits qui changent de position. En effet, il est inutile d'utiliser de l'espace mémoire pour un bit qui ne change pas de position. Il est à noter que les notions de délai et de mémoire sont liées. La mémoire requise est au moins le délai requis.

4.8.3 Facteur d'étalement

Le facteur d'étalement est un des facteurs les plus représentatifs de la capacité d'un entrelaceur. Il permet de visualiser la densité des bits après entrelacement. Il s'agit tout simplement de dresser un graphique illustrant la position des bits après entrelacement. Grâce à ce facteur, nous pouvons immédiatement conclure sur la corrélation des bits après entrelacement. Étant donné que pour les Codes Turbo, la corrélation joue un grand rôle pour le décodage, dresser le graphe du facteur d'étalement de l'entrelaceur que nous utilisons est un outil primordial pour juger des performances que nous obtenons.

Aux figures 4.9 et 4.10 sont illustrés deux exemples de facteur d'étalement pour deux entrelaceurs différents. Nous observons, à la figure 4.9 qu'il existe deux zones sur le graphe non occupées par les bits après entrelacement. Cet entrelaceur n'étale donc pas de façon adéquate les bits. Ces derniers sont bien trop rapprochés les uns des autres. Ceci a pour effet de créer une certaine corrélation entre eux. À l'inverse, à la figure 4.10, l'entrelaceur utilise tout l'espace disponible dans le graphe pour étaler les bits de sortie. Ce dernier a donc toutes les chances de décorréler les bits. Nous voyons bien que la densité des bits est moins importante qu'à la figure 4.9. Toutefois, il est important de noter que ce genre de graphe illustre la puissance de l'entrelaceur de séparer les bits. Il ne donne en aucun cas la performance des Codes Turbo. Il existe beaucoup de paramètres à prendre en compte et l'entrelaceur n'en est qu'un. Néanmoins, ce facteur d'étalement est un excellent moyen de pouvoir interpréter les performances de nos simulations.

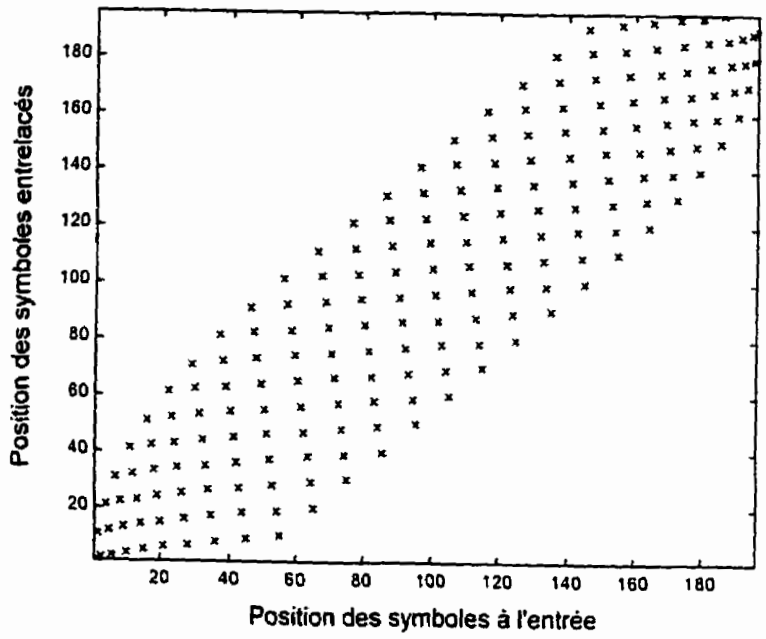


Figure 4.9 – Un entrelaceur dont le facteur d'étalement est mauvais

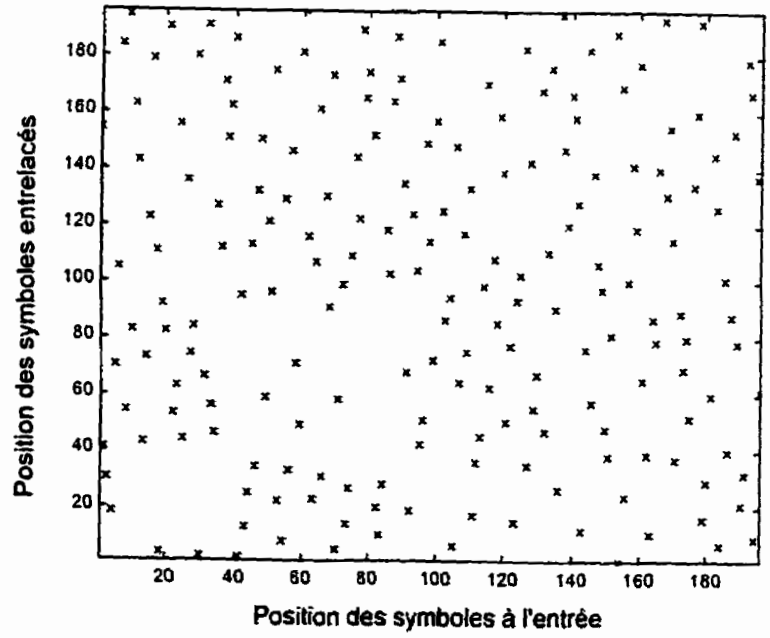


Figure 4.10 – Un entrelaceur dont le facteur d'étalement est bon

4.9 Conclusion

Dans ce chapitre, nous avons donc étudié deux notions très importantes des Codes Turbo. Dans un premier temps, la notion de décodage itératif a été introduite, pour ensuite nous concentrer sur le décodeur turbo. Nous avons ainsi pu déterminer que la puissance des Codes Turbo provient surtout de l'aspect de décodage itératif. Par la suite, nous avons pu nous concentrer sur un des éléments importants des Codes Turbo, à savoir l'entrelaceur. Ce dernier est un mélangeur de bits qui permet de décorréler l'information, vue du décodage turbo. Les paramètres des entrelaceurs ont été étudiés. Nous avons ainsi vu la notion de délai, celle de mémoire, mais surtout le facteur d'étalement qui sera une notion primordiale dans notre étude des entrelaceurs pour les Codes Turbo. Dans notre prochain chapitre, nous allons présenter une première sorte d'entrelaceurs, à savoir les entrelaceurs pseudo-aléatoires.

Chapitre 5 - Entrelaceurs pseudo-aléatoires

Les entrelaceurs pseudo-aléatoires sont des algorithmes dont il est impossible de connaître le développement à cause de la notion d'aléatoire. Lorsque les Codes Turbo ont été découverts, deux types d'entrelaceurs étaient utilisés, à savoir les entrelaceurs blocs et les entrelaceurs pseudo-aléatoires. Ces derniers ont rapidement donné de meilleurs résultats que les premiers. Ces meilleurs résultats furent, entre autres, justifiés par ce caractère aléatoire de l'entrelaceur, capable de mieux répartir l'information extrinsèque au décodage. À partir de ces résultats, certains chercheurs ont mis l'accent sur l'aléatoire et ont décidé de développer des entrelaceurs meilleurs que les pseudo-aléatoires tout en gardant l'aspect de hasard [28]. C'est ainsi que les entrelaceurs symétriques S ont été introduits. Ces derniers, nous le verrons, donnent de très bons résultats et même souvent de meilleurs résultats que les pseudo-aléatoires. Dans ce chapitre, nous allons donc introduire les entrelaceurs pseudo-aléatoires pour ensuite nous concentrer sur les symétriques S.

5.1 Les entrelaceurs pseudo-aléatoires

5.1.1 Principe d'aléatoire

Le concept d'entrelaceurs pseudo-aléatoires est extrêmement simple. Il s'agit tout simplement de générer, aléatoirement, une nouvelle séquence d'ordre pour les bits. En général, l'aléatoire est crée grâce à une racine car cette notion n'existe pas pour les ordinateurs. En fait, il n'existe pas vraiment d'algorithme permettant de créer un aléatoire parfait. Par aléatoire parfait, nous entendons un algorithme qui, répété infiniment, ne redonnera jamais la même séquence. Informatiquement, nous devons donc générer un algorithme aléatoire grâce à la racine. De façon pratique, pour dé-entrelacer, il s'agira de transmettre cette racine ou alors de transmettre une table. Dans tous les cas, la génération

d'un tel algorithme n'est pas aisée. Néanmoins, comme nous allons le voir, ce type d'entrelaceur donne de très bons résultats.

5.1.2 Résultats

Dans le chapitre précédent, nous n'avons inclus aucun résultats, alors que suite à la description du concept des Codes Turbo, il aurait été logique d'en présenter. Néanmoins, comme nous avons décidé d'expliquer ces résultats au moyen des entrelaceurs aléatoires, nous les présentons maintenant.

Comme nous l'avons déjà vu, les Codes Turbo sont surtout caractérisés par le taux de codage et la longueur de contrainte. Le but de ce paragraphe est de montrer la capacité de correction des erreurs des Codes Turbo. Il est important de noter que les simulations sont pour un canal AWGN. De surplus, nous n'avons pas utilisé de queue pour la terminaison des treillis. Les paramètres des Codes Turbo utilisés varient et sont indiqués en titre sur les figures. Toutefois, en ce qui concerne la longueur de contrainte, nous avons utilisé K = 3 et K = 5. Le taux de codage global de ces codes est 1/3 ou 1/2. L'évaluation des performances se fait par le calcul de la probabilité d'erreur par bit (BER) en fonction du E_b/N_0. Le nombre d'itérations que nous avons utilisées est égal à 5. Enfin, l'algorithme log-MAP a été utilisé.

5.1.2.1 Codes Turbo $R_t = 1/3$ et $K = 3$

Les figures 5.1 à 5.4 illustrent des Codes Turbo de taux 1/3. La taille des blocs varie entre 196 et 3600 symboles. À partir de ces figures, nous pouvons constater que pour des faibles rapports signal sur bruit, le probabilité d'erreur s'améliore avec la taille du bloc. Cette dernière correspond à la taille de l'entrelaceur. L'amélioration constatée est de l'ordre de 1/N, N étant la taille des blocs. En effet, en prenant la troisième itération, à 1.5 dB, pour N = 400, nous avons une BER d'environ 10^{-3} et pour N = 3600 de 9.10^{-5}, ce

qui nous fait un rapport d'environ 11, qui est du même ordre de grandeur que 3600 / 400 = 9. Cette affirmation a été démontrée par Benedetto [5] pour les entrelaceurs blocs. En effet, ce dernier a démontré que la probabilité d'erreur par bit était bornée par :

$$P_b \leq \sum_{k=1}^{\lfloor N/2 \rfloor} 2k \binom{2k}{k} \frac{1}{N} \frac{\left(H^{2+2z_{min}}\right)^k}{\left(1-H^{2z_{min}-2}\right)^{2k}} \Bigg|_{H=\exp\left(-\frac{RE_b}{N_0}\right)} \qquad (5.1)$$

où z_{min} est le poids minimum des bits de parité dans les événements erreur générés par des séquences d'information de poids 2. D'après cette équation, il apparaît alors que le gain apporté par l'entrelaceur est inversement proportionnel à la longueur N des blocs.

Le nombre d'itérations requis décroît quand le rapport signal sur bruit augmente. En effet, quelle que soit le figure considérée, pour de larges rapports, trois itérations suffisent. Toutefois, pour de plus petits E_b/N_0, cinq itérations sont nécessaires.

Le gain entre deux itérations consécutives décroît avec le nombre d'itérations. En effet, si nous prenons N = 900, à une BER de 10^{-2}, nous constatons que le E_b/N_0 de la première itération est d'environ 2.5 dB. Celui de la deuxième est environ de 1.2 dB, la troisième 0.8 dB, la quatrième 0.7 dB, et la cinquième 0.6 dB. Les gains entre chaque itération sont donc respectivement de 1.3 dB, 0.4 dB, 0.1 dB et 0.1 dB. Entre la première et la deuxième itération, le gain est donc considérable. Entre la deuxième et la troisième, il est encore conséquent. Toutefois, les gains suivants ne sont pas importants. Ceci tend donc à conclure que les Codes Turbo ont tendance à saturer à plus grand E_b/N_0. Ceci amène aussi à penser que dans certains cas, ajouter de la complexité (nombre d'itérations) n'apporte pas toujours un bon gain. Dans le cas des Codes Turbo, il faut donc savoir faire un compromis entre complexité et gain.

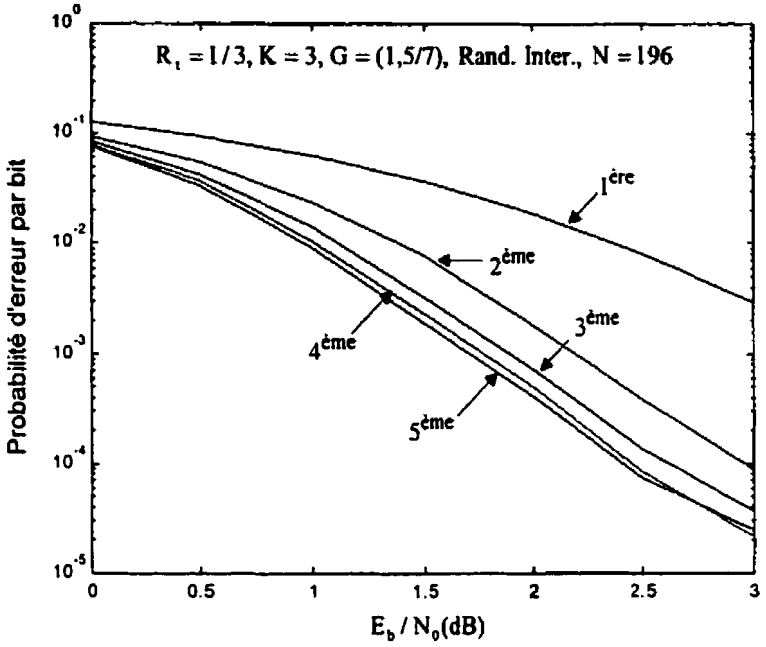


Figure 5.1 – BER, R_t = 1/3, K = 3, entrelaceur aléatoire, canal AWGN, N = 196

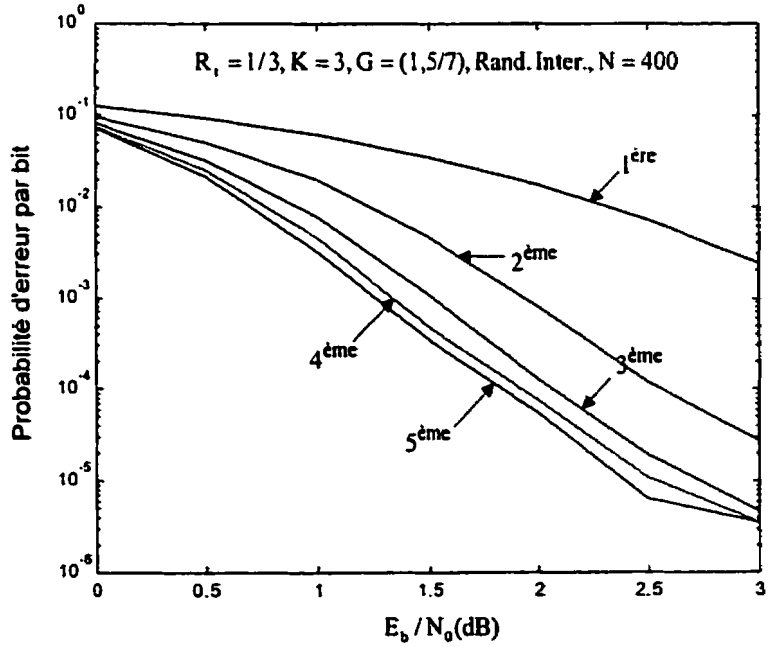


Figure 5.2 – BER, R_t = 1/3, K = 3, entrelaceur aléatoire, canal AWGN, N = 400

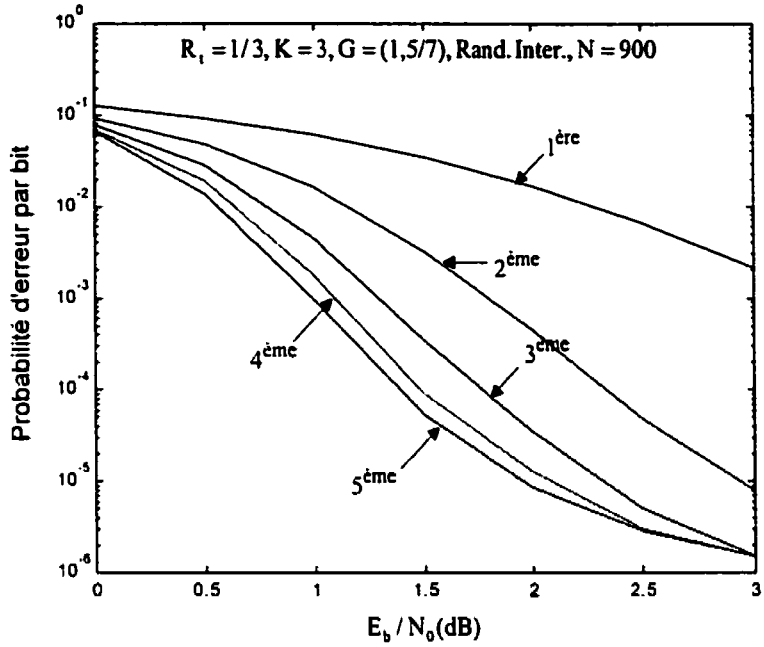


Figure 5.3 – BER, R_t = 1/3, K = 3, entrelaceur aléatoire, canal AWGN, N = 900

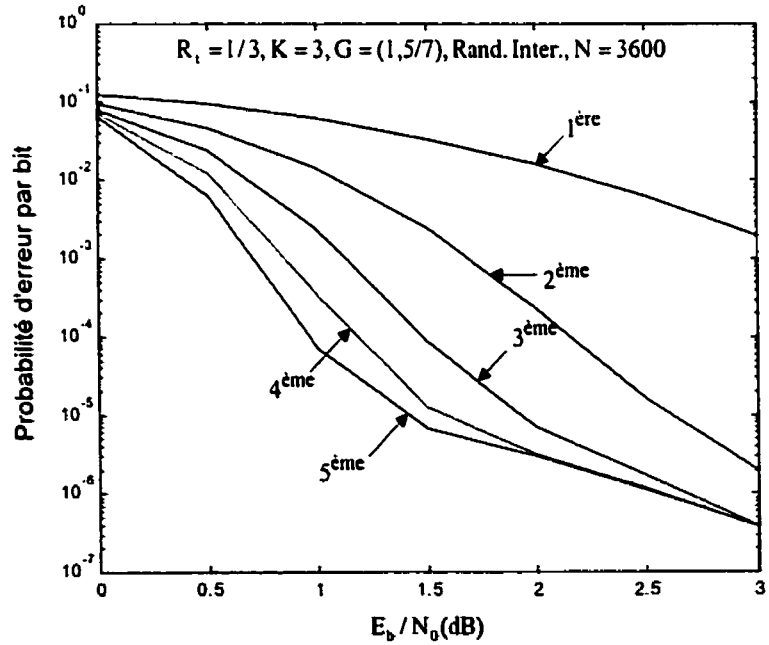


Figure 5.4 – BER, R_t = 1/3, K = 3, entrelaceur aléatoire, canal AWGN, N = 3600

5.1.2.2 Codes Turbo $R_t = 1/2$ et $K = 3$

À titre de comparaison avec le taux de codage 1/3, nous avons inclus les mêmes résultats pour un taux de codage moindre, à savoir $1/2$. Une méthode pour obtenir un tel codage est de perforer un codeur turbo de taux 1/3. La relation entre la taille de l'entrelaceur et la probabilité d'erreur est encore illustrée pour un tel taux. Les remarques que nous avons faites sur les itérations sont aussi identiques. La seule différence avec le taux de codage 1/3 réside dans les performances. Nous constatons que pour un taux de $1/2$, les performances ne sont pas aussi bonnes que pour le taux 1/3. Ceci est normal étant donné que pour un taux $1/2$, nous envoyons moins de bits de parité. Lors du décodage, l'information extrinsèque ne sera donc pas aussi performante que si on envoyait deux bits de parité. Enfin, nous remarquons aussi que la perte par rapport au taux 1/3 décroît quand E_b/N_0 croît.

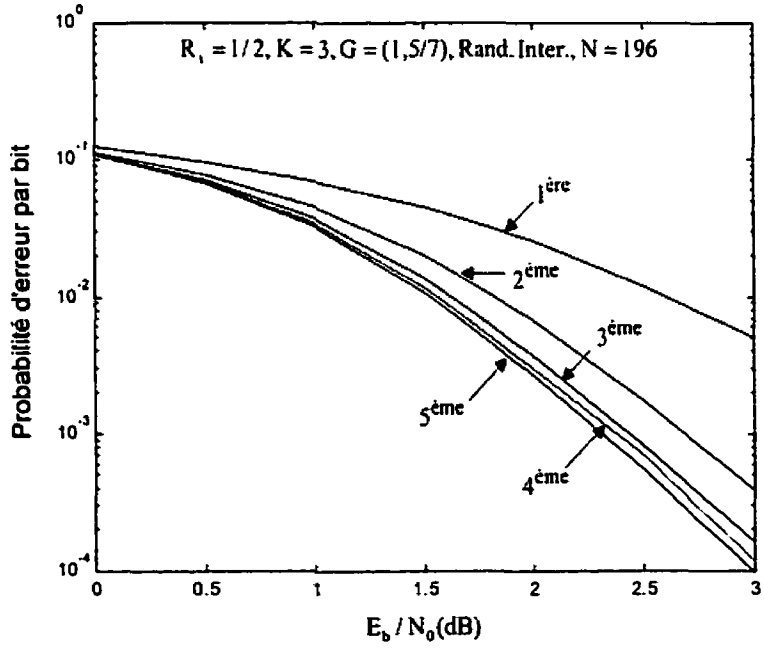


Figure 5.5 – BER, $R_t = 1/2$, $K = 3$, entrelaceur aléatoire, canal AWGN, N = 196

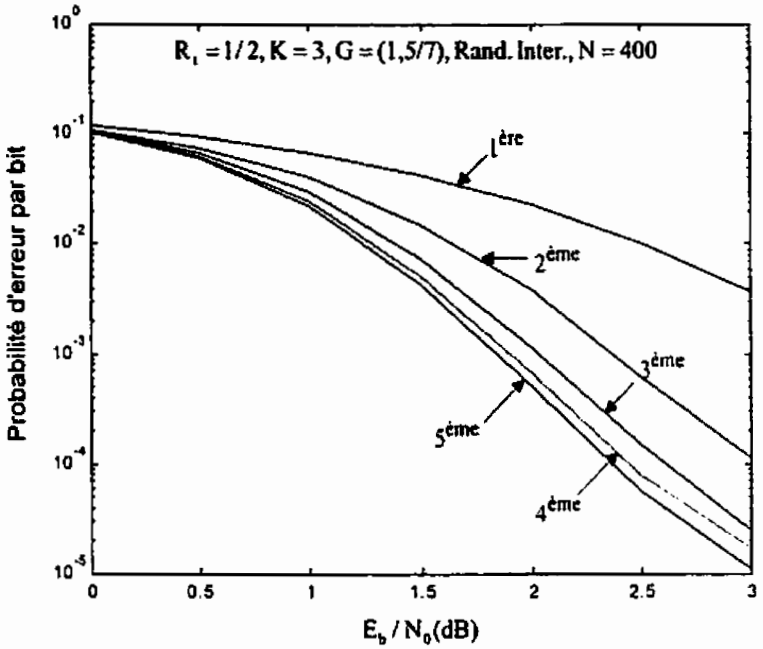


Figure 5.6 – BER, $R_t = 1/2$, K = 3, entrelaceur aléatoire, canal AWGN, N = 400

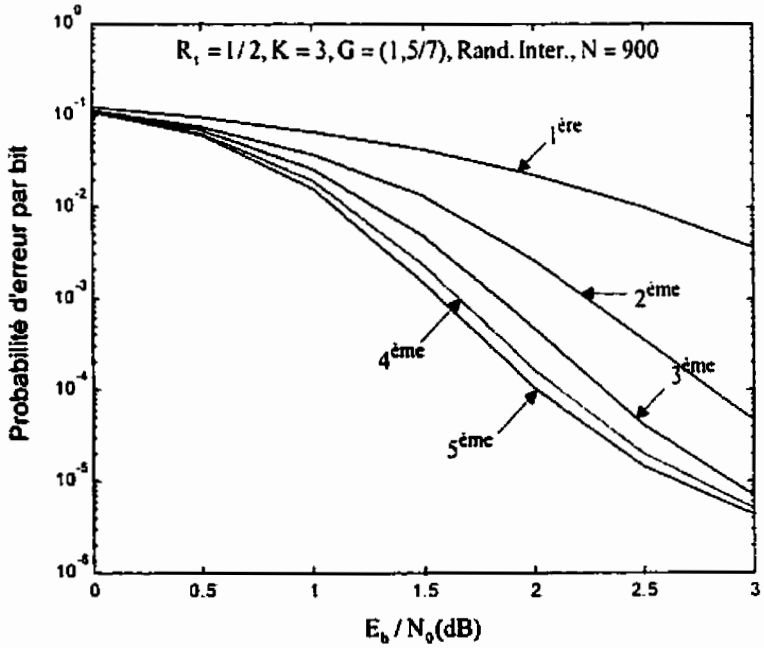


Figure 5.7 – BER, $R_t = 1/2$, K = 3, entrelaceur aléatoire, canal AWGN, N = 900

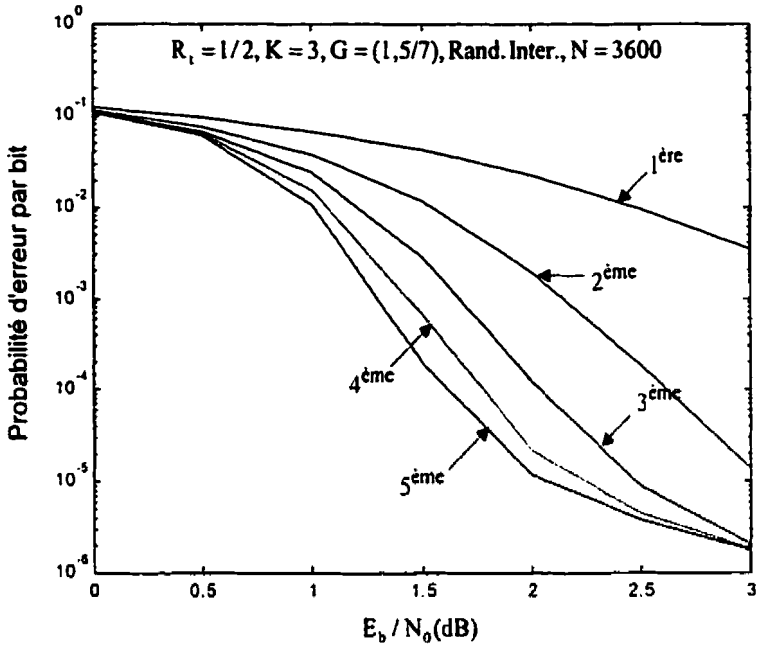


Figure 5.8 – BER, R_t = 1/2, K = 3, entrelaceur aléatoire, canal AWGN, N = 3600

5.1.2.3 Codes Turbo K = 5

Maintenant que nous avons comparé deux taux de codage différents pour les Codes Turbo, il est bon de constater l'influence de la longueur de contrainte sur les performances. Nous avons donc changé cette contrainte et avons pris K = 5. Pour un taux de codage 1/3, pour de faibles rapports signal sur bruit, il semble que la longueur K = 5 n'apporte pas grand chose de plus que K = 3. Aucun gain n'est constaté. Cela n'est donc pas nécessaire de compliquer le processus de codage. Toutefois, nous notons un certain gain pour de plus grands rapports. Les mêmes remarques qu'aux deux paragraphes précédents sont encore valables.

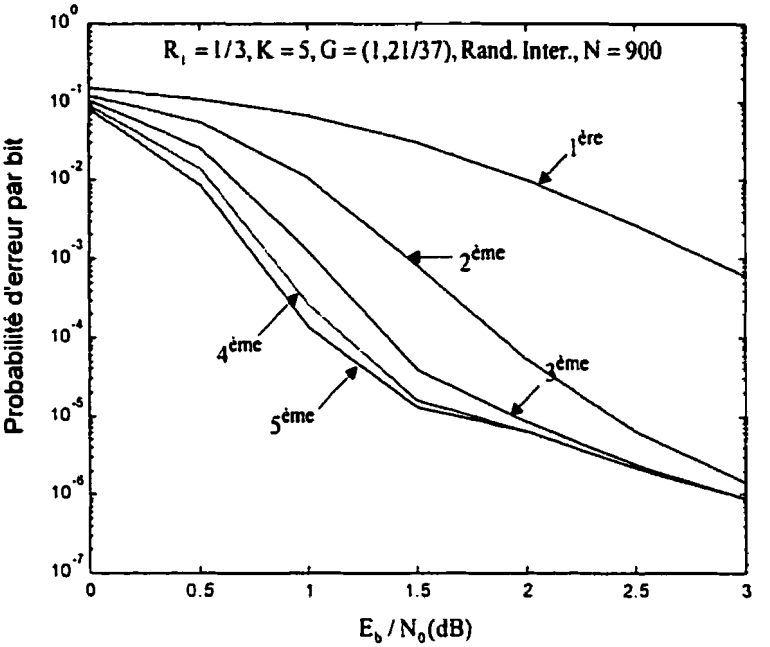


Figure 5.9 – BER, $R_t = 1/3$, K = 5, entrelaceur aléatoire, canal AWGN, N = 900

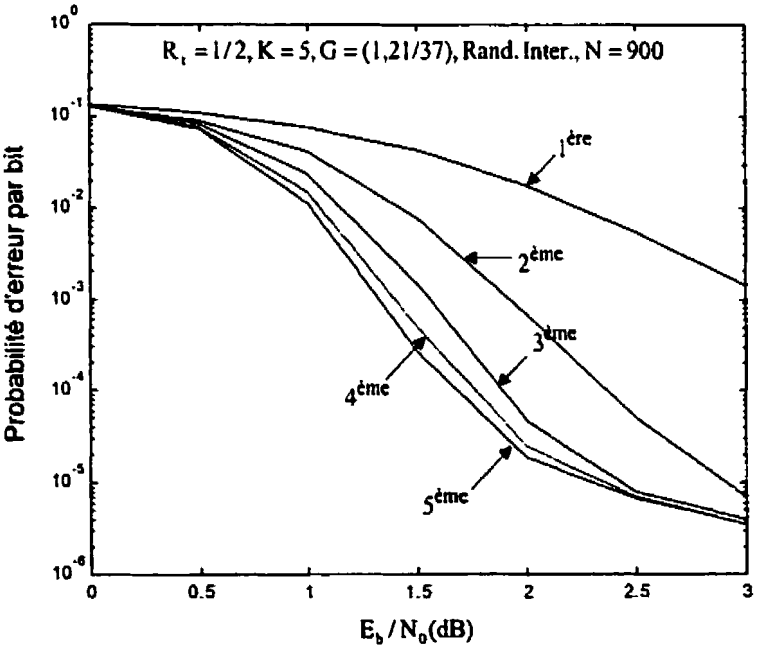


Figure 5.10 – BER, $R_t = 1/2$, K = 5, entrelaceur aléatoire, canal AWGN, N = 900

5.2 Les entrelaceurs symétriques S [28]

5.2.1 Les entrelaceurs symétriques

Le désavantage de la plupart des entrelaceurs réside dans le fait que nous avons besoin d'une séquence d'entrelacement et d'une autre inverse au processus d'entrelacement. En effet, comme ces séquences sont différentes, des tables d'allocation disjointes sont nécessaires. Ceci requiert, entre autres, plus de mémoire que si nous pouvions faire en sorte que ces deux processus soient réduits à un. Néanmoins, ce problème peut être résolu de façon assez simple. En effet, nous pouvons imaginer un processus de mélange qui serait identique à l'entrelacement et au dé-entrelacement. Ceci peut être effectué en utilisant une opération de symétrie.

Un entrelaceur symétrique échange la position de deux bits entre eux. Ceci veut dire que si le bit 9 après entrelacement se retrouve à la position 5, alors obligatoirement, le bit 5 se retrouvera à la position 9. Nous avons illustré un entrelaceur symétrique à la figure 5.12. Nous avons également, à titre de comparaison, illustré un entrelaceur non symétrique. Les séquences d'entrelacement et de dé-entrelacement sont indiquées. Nous constatons, dans le cas de l'entrelacement symétrique, que ces deux séquences sont identiques.

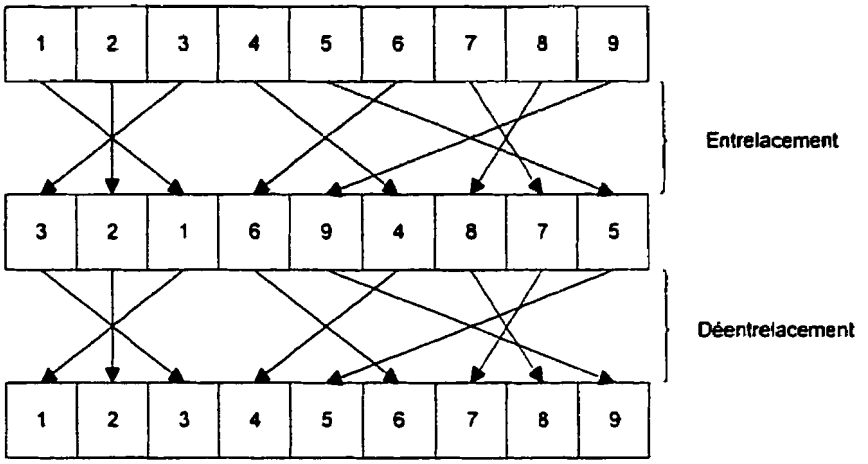


Figure 5.11 – Un processus d'entrelacement symétrique

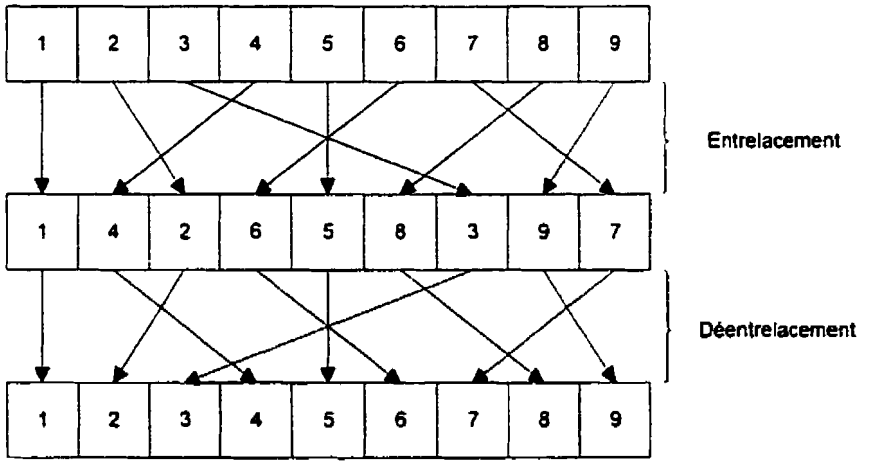


Figure 5.12 – Un processus d'entrelacement non symétrique

5.2.2 Les entrelaceurs symétriques S

Cet aspect de symétrie est fort intéressant. Toutefois, en utilisant ceci, nous perdons un peu de la propriété d'aléatoire requise pour le décodage turbo de façon à décorréler les informations transmises entre les décodeurs. Une idée intéressante serait alors d'imposer cet aléatoire dans un algorithme d'entrelacement symétrique. Une contrainte supplémentaire est ensuite d'ajouter à cet algorithme un aspect de distance. Il s'agit de

forcer les bits entrelacés d'être distants de leurs bits voisins d'origine. Par exemple, nous pourrons implanter un algorithme d'entrelacement symétrique de longueur N = 196 avec une distance de 3. La distance de cet entrelaceur est un paramètre important et il est noté S. L'algorithme consiste alors à générer deux bits aléatoires, de vérifier qu'ils sont bien distants de S et de les permuter pour obtenir la symétrie. L'algorithme d'entrelacement symétrique S est alors :

1. Générer un tableau de longueur N;
2. Générer deux nombres au hasard qui n'ont pas été au préalable insérés dans le tableau;
3. Si la distance entre ces deux nombres est inférieure à S, retourner à l'étape 2;
4. Si la distance entre ces deux nombres et les S nombres précédents ou suivants dans le tableau est inférieure à S, retourner à l'étape 2, sinon, interchanger ces deux nombres;
5. Si le tableau n'est pas plein, retourner à l'étape 2;
6. Si le tableau est complet, alors, arrêter, sinon, retourner à l'étape 1.

Cet algorithme est illustré à la figure 5.13. Il est important de noter qu'un tel algorithme peut s'avérer très complexe si la distance est grande. De plus, il est évident qu'il existe une séparation maximale possible pour un N donné. Par exemple, il est impossible de générer un tel algorithme avec N = 196 et S = 50.

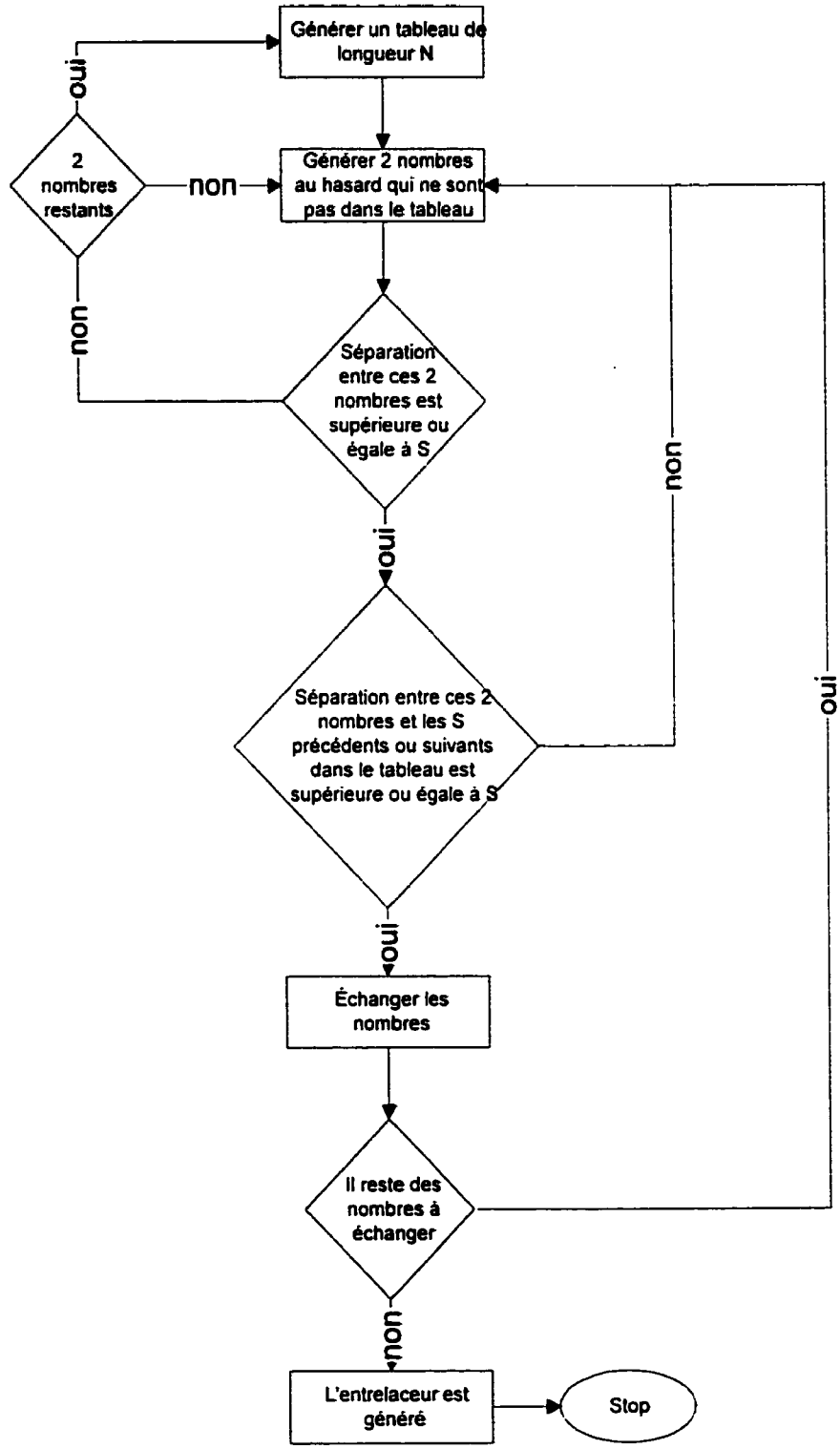


Figure 5.13 – Algorithme de l'entrelaceur symétrique S

5.2.3 Résultats

Nous avons effectué des simulations pour évaluer la qualité de l'entrelaceur symétrique S. Nous avons simulé un codeur turbo de taux global 1/3, de longueur de contrainte 3, dans un canal AWGN. L'entrelaceur symétrique S possède deux paramètres variables. Tout comme tout entrelaceur, sa longueur peut varier. Le paramètre S peut également lui changer. Nous avons donc effectué nos simulations pour des longueurs d'entrelaceurs entre 196 bits et 900 bits. Quant au paramètre S, plus la longueur de l'entrelaceur est élevée, plus nous pouvons l'augmenter. Nous allons, dans ce paragraphe, présenter nos résultats.

5.2.3.1 Entrelaceurs de longueur 196 bits

Pour ce type d'entrelaceur, le domaine d'ensemble des S possibles est assez restreint. Nous avons été capables de simuler jusqu'à S = 10. De surplus, pour S = 10, l'algorithme que nous avons programmé a mis un temps important à développer la séquence d'entrelacement. Nous présentons deux résultats de simulations aux figures 5.14 et 5.15.

Globalement, les performances de ce type d'entrelaceur sont excellentes, même pour de petits blocs comme 196 bits. Ces performances seront à comparer aux autres types d'entrelaceurs plus classiques et moins complexes pour des petits blocs. Nous verrons alors s'il est intéressant de payer le prix de la complexité. En effet, plus S est grand, plus l'algorithme de cet entrelaceur est complexe. En comparant rapidement avec l'entrelaceur aléatoire, à la troisième itération, pour une BER de 10^{-4} à la figure 5.1, nous obtenons un E_b/N_0 de 2.5 dB. Pour le symétrique S = 3, nous obtenons déjà 2.4 dB, ce qui nous procure un gain de 0.1 dB, rien qu'en forçant une séparation minime. Quant à la séparation S = 7, nous obtenons 2.1 dB environ, ce qui procure un gain de 0.4 dB. Nous le voyons donc, une certaine amélioration est apportée avec cet entrelaceur. Notons que

ces gains sont obtenus pour des petits blocs. Nous verrons, par la suite, les gains de blocs plus importants.

Nous notons un bon gain entre les différentes itérations. Ceci est principalement dû au fait que cet algorithme est un dérivé de l'entrelacement aléatoire. Il est donc normal de noter le même comportement. Toutefois, dans le cas de l'entrelacement aléatoire, nous avons noté que les Codes Turbo tendaient à saturer au-delà d'une certaine valeur de E_b/N_0. Il semblerait, en regardant les figures 5.14 et 5.15 que ce ne soit pas le cas pour les entrelaceurs symétriques S.

Nous observons un très bon gain de la première à la troisième itération, tout comme pour le cas de l'entrelaceur aléatoire. Toutefois, à partir de la troisième itération, le gain observé est encore plus minime que dans le cas de l'entrelacement aléatoire. Pour une BER de 10^{-4}, le gain entre la deuxième et la troisième itération est d'environ 0.5 dB pour S = 7. Il n'est alors plus que de 0.1 dB entre la troisième et la quatrième. Ceci tend donc à confirmer que l'aspect de symétrie ainsi que la séparation imposée apportent une nette amélioration pour un petit nombre d'itérations. La complexité que nous ajoutons dans l'algorithme nous permet donc de diminuer le nombre d'itérations pour avoir les mêmes résultats.

En comparant les performances pour S = 3 et S = 7, il vient qu'elles sont bien meilleures pour un paramètre de 7. Aux figures 5.16 et 5.17, nous avons tracé la courbe du facteur d'étalement de ces deux entrelaceurs. Nous observons que l'entrelaceur S = 7 possède un étalement plus important que le S = 3. Ceci est logique étant donné que nous avons imposé une distance de 7 au lieu de 3. En effet, à la figure 5.16, pour S = 3, nous constatons que de nombreux bits sont proches les uns des autres. Ces derniers seront alors plus ou moins corrélés entre eux. Ceci tend donc à conclure que plus S augmente, meilleures sont les performances. Toutefois, nous verrons dans le prochain paragraphe que ce n'est pas toujours le cas. En fait, il existe une limite pour le facteur S en ce qui

concerne les performances. Il est à noter que nous aurions pu avoir les mêmes performances pour le cas S = 3. En effet, selon l'algorithme, il est tout à fait possible qu'en obligeant une séparation de 3, nous nous retrouvions avec une séparation de 7. Toutefois, la probabilité que ceci soit le cas est faible.

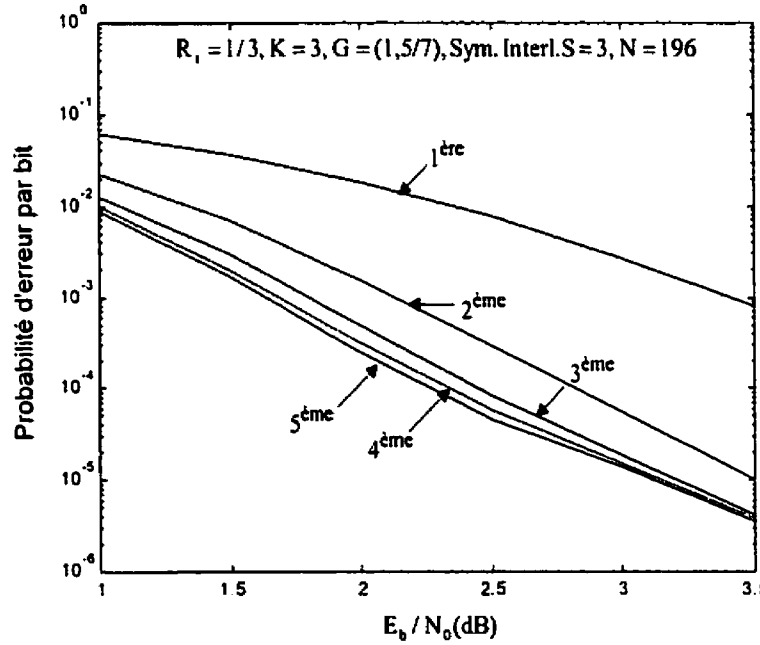


Figure 5.14 – BER, R_t = 1/3, K = 3, ent. symétrique (3), canal AWGN, N = 196

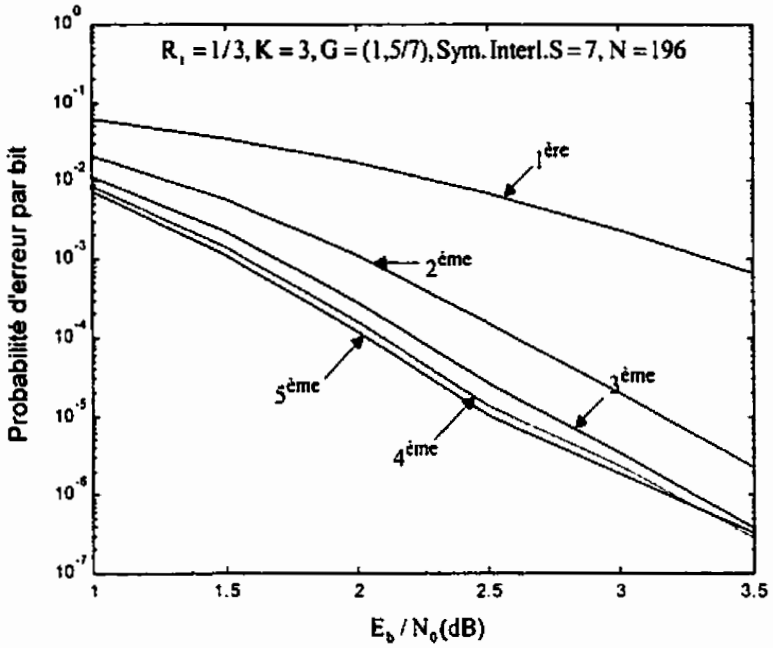


Figure 5.15 – BER, R_t = 1/3, K = 3, ent. symétrique (7), canal AWGN, N = 196

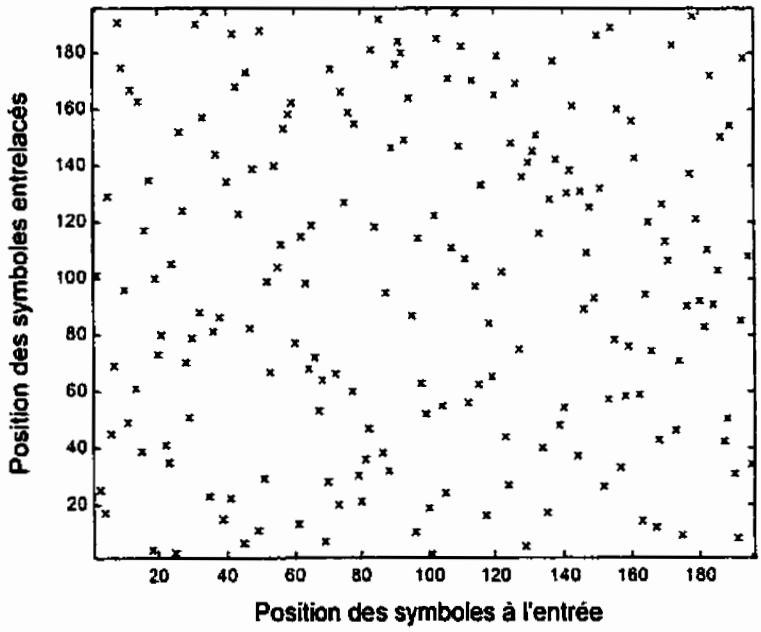


Figure 5.16 – Facteur d'étalement, entrelaceur symétrique S = 3, N = 196

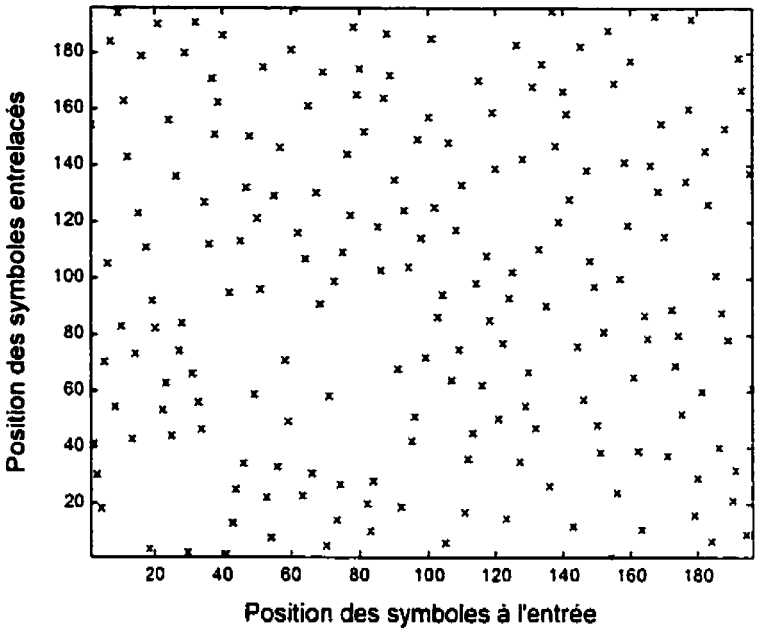


Figure 5.17 – Facteur d'étalement, entrelaceur symétrique S = 7, N = 196

5.2.3.2 Entrelaceurs de longueur 400 bits

Nous venons donc de voir que les performances dans le cas de 196 bits semblent excellentes. Il est donc prévisible qu'elles le seront également pour des longueurs supérieures. Nous avons donc effectué des simulations pour N = 400. Dans ce cas, l'ensemble des S possibles est plus important. Nous avons simulé jusqu'à S = 14. Nous présentons deux simulations aux figures 5.18 et 5.19.

En comparant avec l'entrelacement aléatoire pour 400 bits, nous avons, toujours pour la troisième itération à 2.5 dB un gain à 10^{-4} un gain de 0.2 dB pour S = 3 et de 0.4 dB pour S = 10 par rapport à l'entrelacement aléatoire. L'entrelacement symétrique S est donc meilleur pour N = 400 encore. Notons que pour cette longueur, le meilleur entrelacement s'effectue pour S = 10.

Nous notons toujours un bon gain entre les différentes itérations. Les Codes Turbo semblent encore ne pas saturer dans ce cas. Les gains entre chaque itération sont encore importants jusqu'à la troisième. Ensuite, nous avons des gains minimes.

En comparant les performances pour S = 3 et S = 10, il apparaît qu'elles sont bien meilleures pour un paramètre de 10. Aux figures 5.18 et 5.19, nous avons encore tracé la courbe du facteur d'étalement de ces deux entrelaceurs. Encore une fois, nous observons une nette amélioration pour un S important.

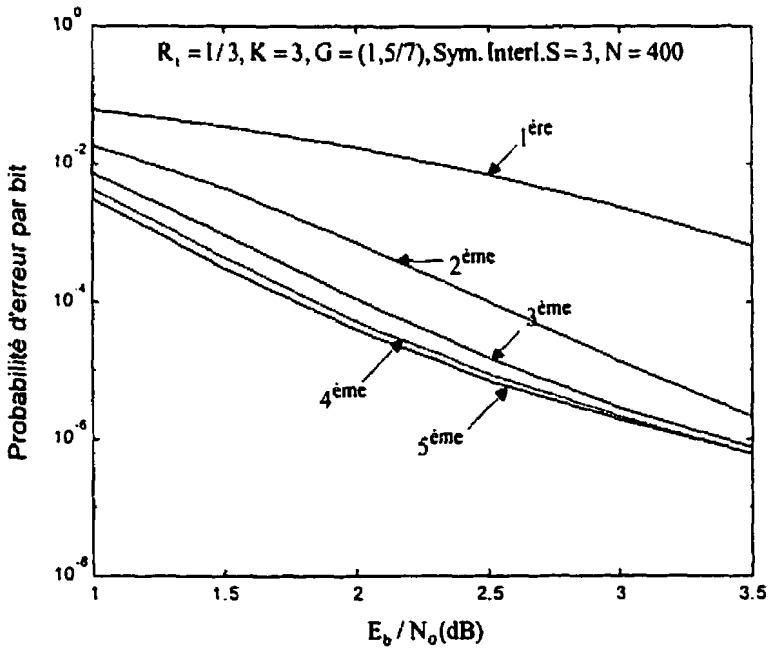


Figure 5.18 – BER, R_t = 1/3, K = 3, ent. symétrique (3), canal AWGN, N = 400

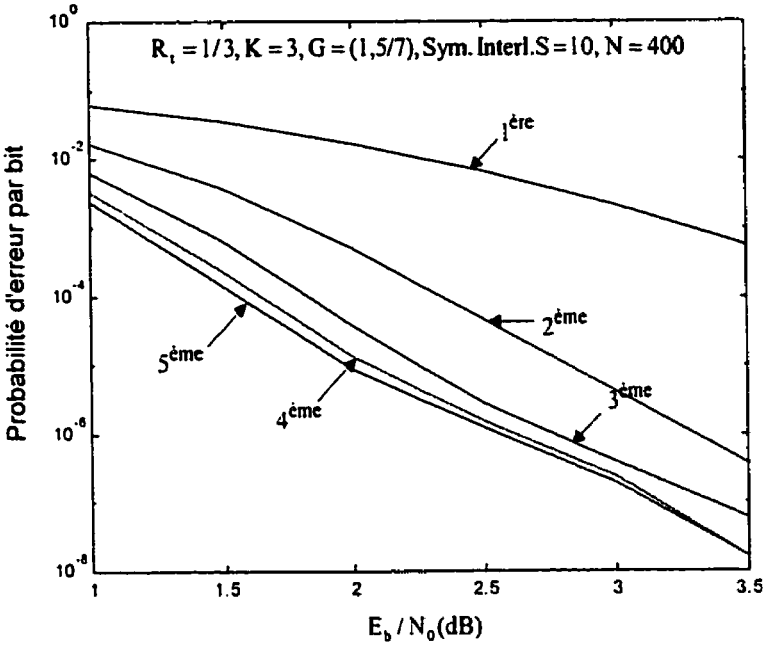


Figure 5.19 – BER, $R_t = 1/3$, K = 3, ent. symétrique (10), canal AWGN, N = 400

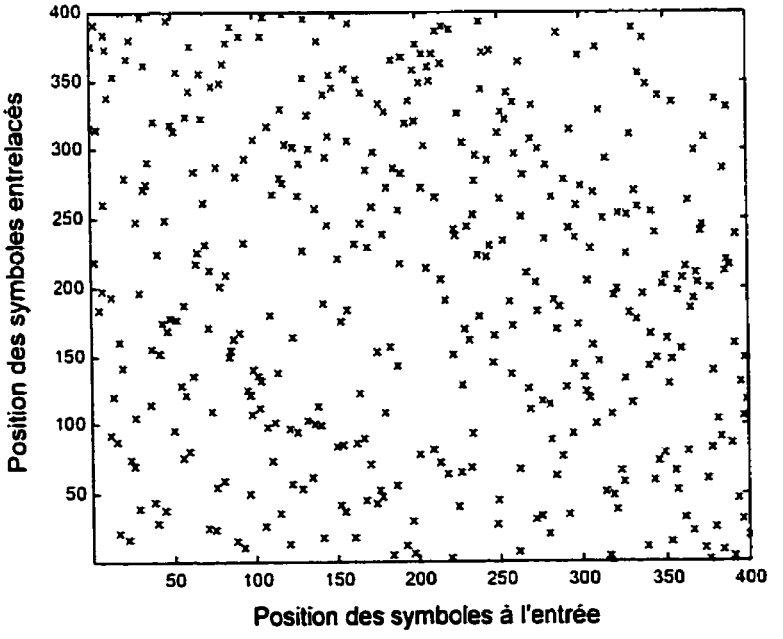


Figure 5.20 – Facteur d'étalement, entrelaceur symétrique S = 3, N = 400

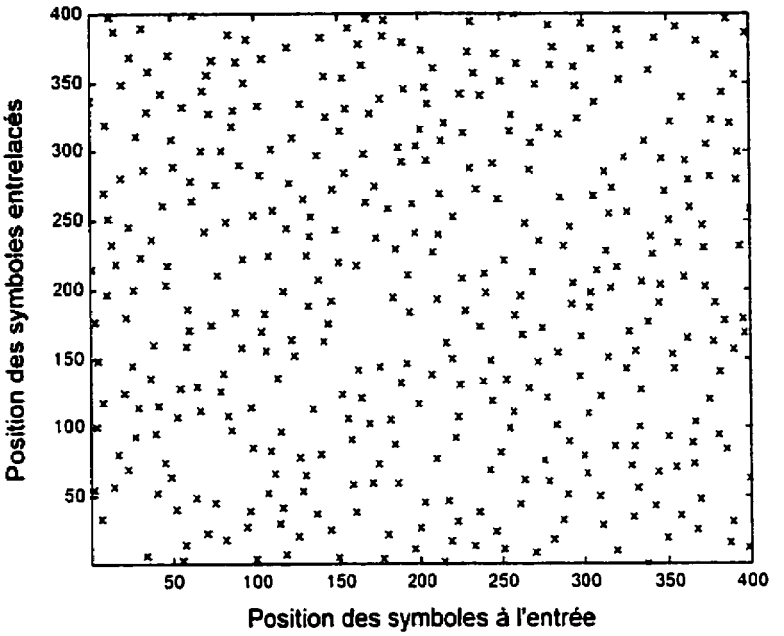


Figure 5.21 – Facteur d'étalement, entrelaceur symétrique S = 10, N = 400

5.2.3.2 Entrelaceurs de longueur 900 bits

La même étude a été effectuée pour des tailles de bloc de 900 bits. Les mêmes conclusions ressortent des figures 5.22, 5.23, 5.24 et 5.25. Cette fois, nous avons été capables de simuler jusqu'à S = 18. Il est à noter une certaine différence entre le cas S = 3 et le cas S = 15. En effet, alors que le premier tend à montrer une saturation à la troisième itération, le deuxième montre que pour un S = 15, nous pouvons obtenir un gain jusqu'à la cinquième itération. Cet entrelaceur est donc capable de corriger très bien les erreurs à mesure que la taille des blocs augmente.

Enfin, les figures 5.24 et 5.25 montrent la différence des facteurs d'étalement pour les cas S = 3 et S = 15. Alors que la première figure montre quelques zones de forte densité, il en est tout autrement pour la deuxième. Le facteur d'étalement du cas S = 15 est vraiment très bon et offre une excellente décorrélation entre les bits.

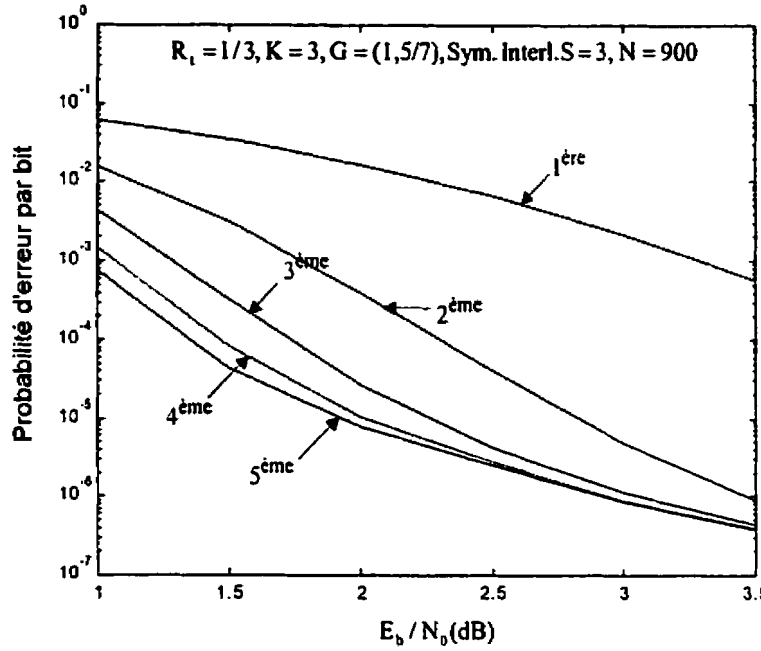


Figure 5.22 – BER, R_t = 1/3, K = 3, ent. symétrique (3), canal AWGN, N = 900

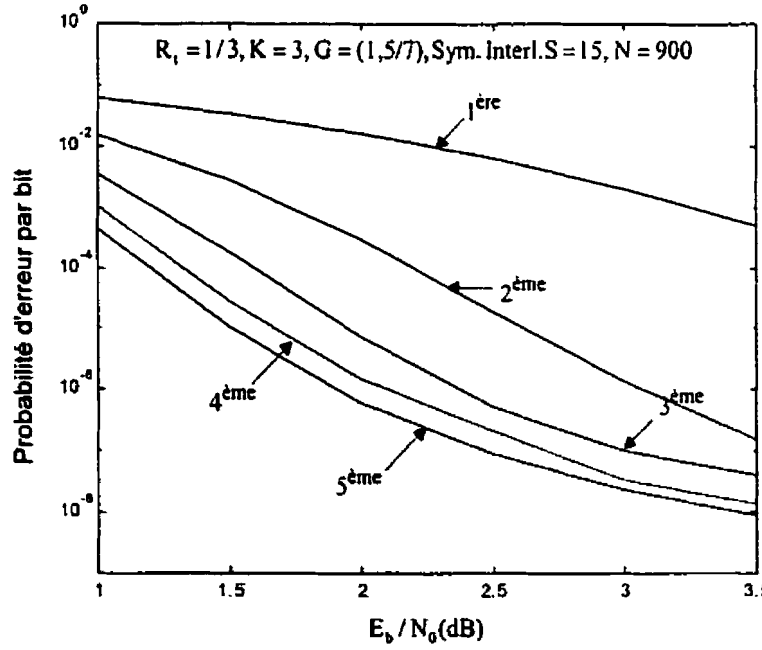


Figure 5.23 – BER, R_t = 1/3, K = 3, ent. symétrique (15), canal AWGN, N = 900

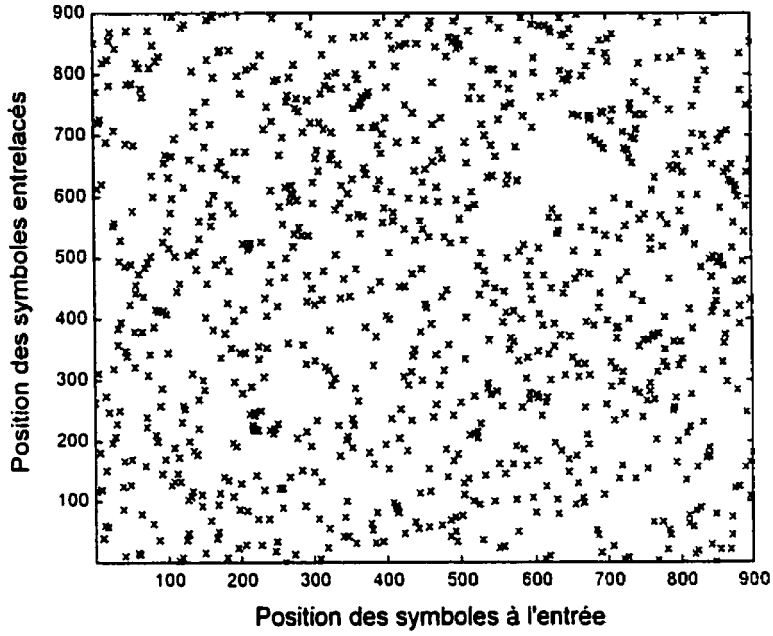


Figure 5.24 – Facteur d'étalement, entrelaceur symétrique S = 3, N = 900

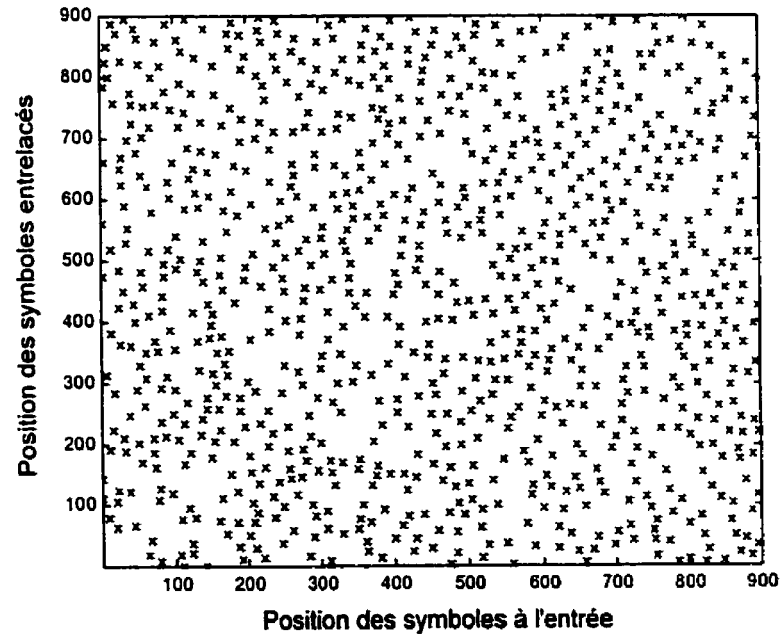


Figure 5.25 – Facteur d'étalement, entrelaceur symétrique S = 15, N = 900

5.2.3.3 Influence du paramètre S

Nous avons mené une petite étude pour ce qui est de l'influence du paramètre S sur les performances des Codes Turbo. Aux figures 5.27, 5.28 et 5.29 se trouvent trois graphiques de comparaison des performances des Codes Turbo en fonction du facteur S à 2.5 dB pour les itérations deux à quatre.

Il ressort un comportement général de ces entrelaceurs. En effet, nous observons un comportement en "U" pour chaque cas. Les performances s'améliorent jusqu'à une certaine valeur de S et tendent à ne plus apporter de gain. Il s'avère même qu'après cette valeur de S, les performances sont réduites. Ceci tend donc à montrer qu'il existe une valeur de S donnée pour laquelle les performances sont les meilleures. Ceci est observé pour toutes les itérations étudiées. De plus, pour toutes les itérations, le valeur idéale de S est la même. Une justification de cette observation serait qu'à partir d'un certain S, la séparation apportée en plus par tous les S supérieures n'améliore pas la décorrélation.

Nous observons respectivement les valeurs optimales de S comme étant 7 pour 196 bits, 10 pour 400 bits et 15 pour 900 bits. Nous pouvons alors émettre la conclusion que les entrelaceurs symétriques S donnent les meilleures performances pour $S = \sqrt{\frac{N}{4}}$.

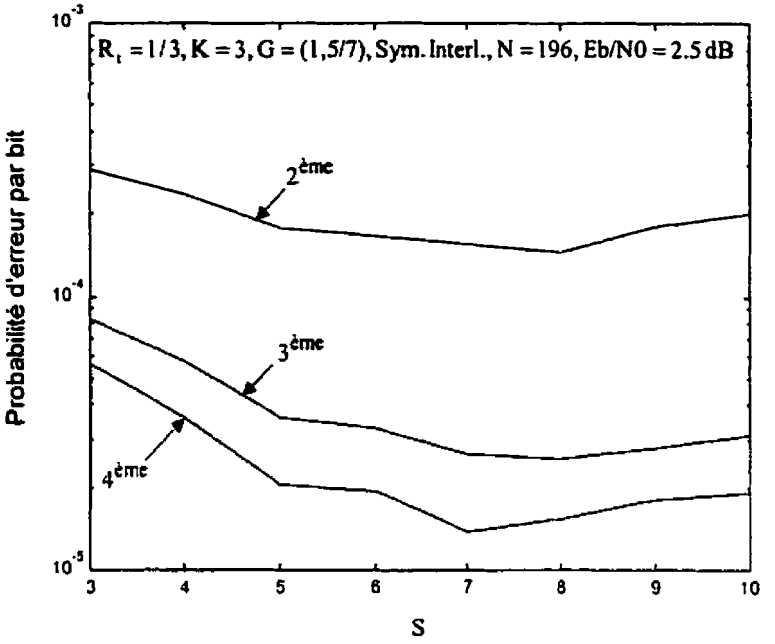


Figure 5.26 – Comparaison des entrelaceurs symétriques, N = 196, 2.5 dB

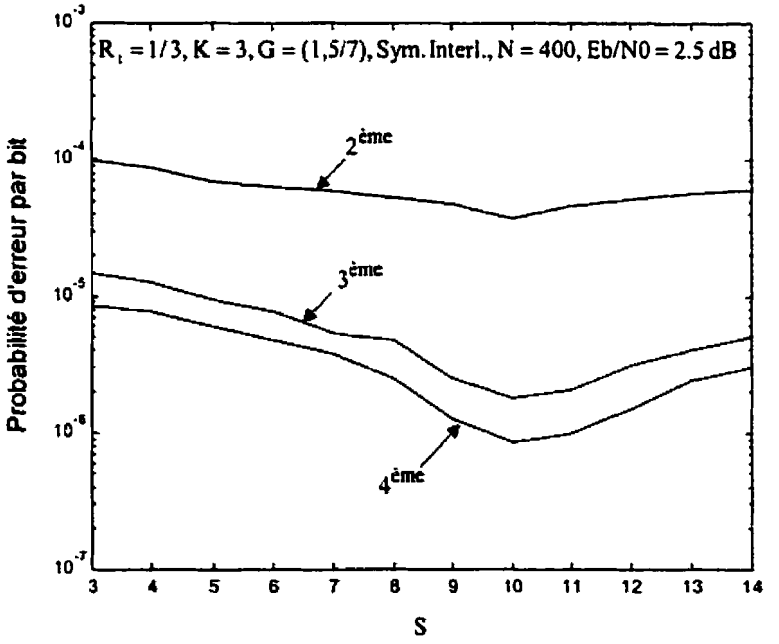


Figure 5.27 – Comparaison des entrelaceurs symétriques, N = 400, 2.5 dB

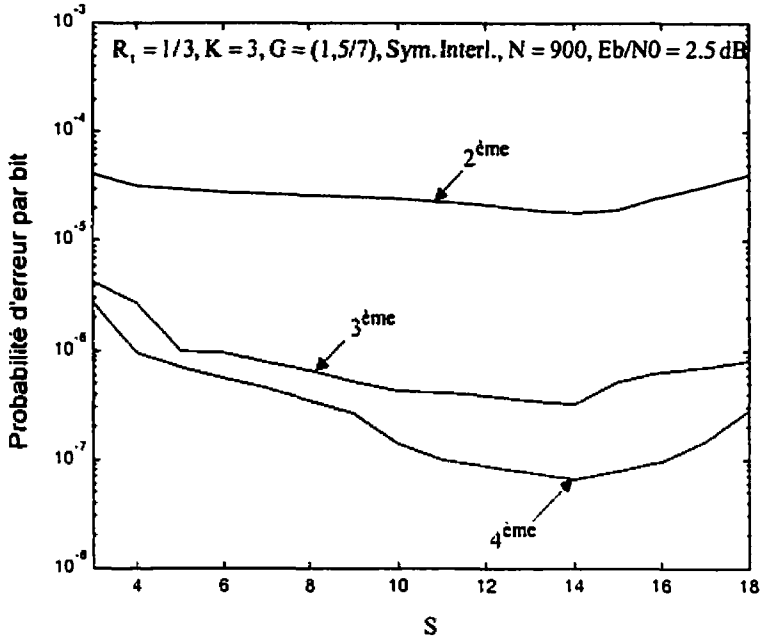


Figure 5.28 – Comparaison des entrelaceurs symétriques, N = 900, 2.5 dB

5.2.3.4 Délai

Nous allons terminer notre étude des entrelaceurs symétriques S par une évaluation en terme d'itérations du délai induit par la génération d'un tel algorithme. Nous sommes sûrs qu'augmenter S implique un délai plus important, mais jusqu'à maintenant, nous n'avons pas évalué ce délai. Nous avons donc, lors de nos simulations, calculé le nombre d'itérations nécessaires et avons tracé les graphes du nombre d'itérations en fonction de S aux figures 5.29, 5.30 et 5.31. De ces trois figures, nous observons immédiatement que ces courbes sont des exponentielles. Plus nous augmentons S, plus grand est le nombre d'itérations. Nous nous attendions à ce résultat étant donné que plus S augmente, plus nous approchons de la limite possible pour la génération de l'algorithme.

Ces graphiques apportent des renseignements très intéressants pour le choix du facteur S. En effet, grâce à ces derniers, nous sommes capables d'évaluer le temps que cela va nous

prendre pour générer l'algorithme. Nous sommes surtout capables de voir la limite en terme de délai de cet entrelaceur. Ainsi, pour N = 196, nous voyons qu'il n'est pas nécessaire d'aller au-delà de S = 7. Ainsi, un bon choix pour 196 bits serait de prendre S = 7. De même, pour N = 400, nous observons que le meilleur choix, en terme de délai raisonnable est S = 10. Enfin, pour N = 900, le meilleur S est 15. Il est fort intéressant de noter que ces limites correspondent exactement aux meilleures performances que nous avons obtenues.

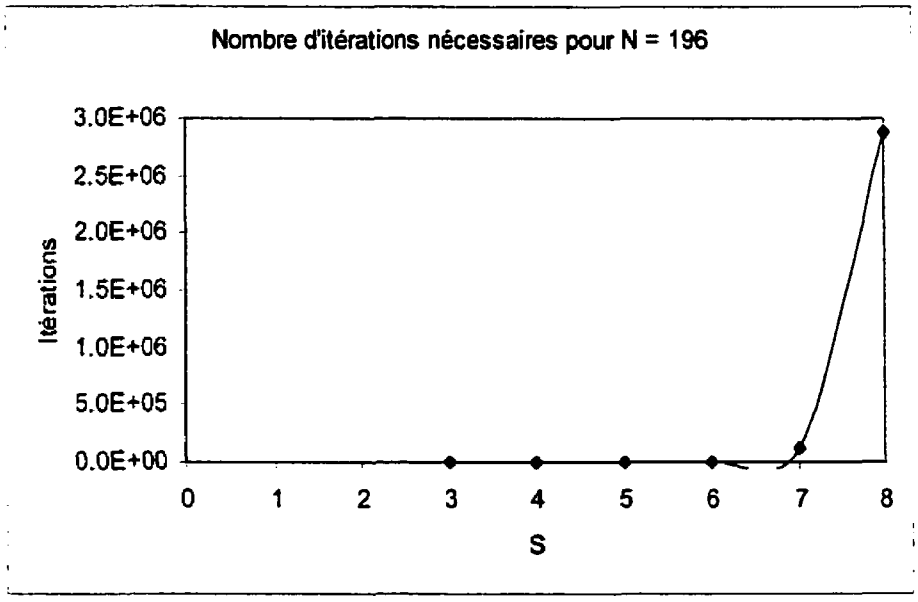


Figure 5.29 – Nombre d'itérations en fonction de S pour N = 196

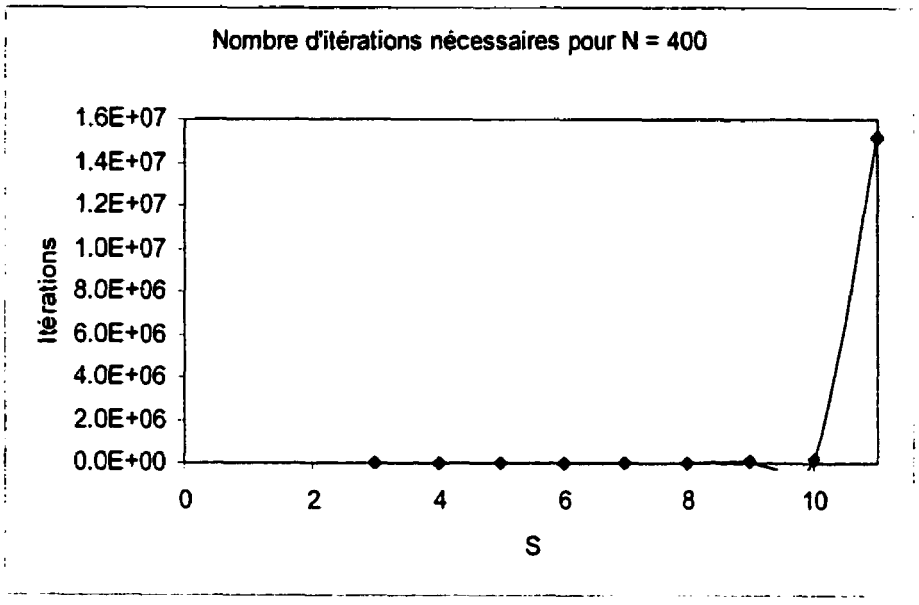


Figure 5.30 – Nombre d'itérations en fonction de S pour N = 400

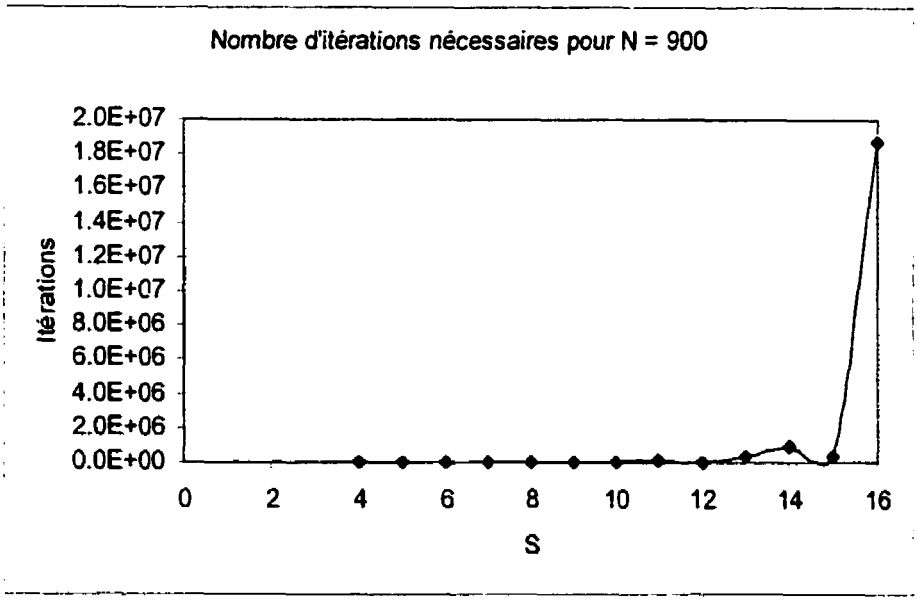


Figure 5.31 – Nombre d'itérations en fonction de S pour N = 900

Chapitre 6 - Entrelaceurs blocs et convolutionnels

Nous allons maintenant continuer notre étude des performances des entrelaceurs avec des systèmes plus classiques. L'intérêt va maintenant être porté sur la simplicité. En effet, même si le besoin de bonnes performances est nécessaire, il n'en va pas moins que souvent, nous n'avons pas besoin d'excellentes performances, mais plutôt de simplicité et rapidité.

Nous allons donc nous intéresser aux entrelaceurs blocs dans un premier temps. Ces derniers font l'objet de beaucoup de recherches et offrent une excellente simplicité. Nous étudierons les performances de ces derniers. Par la suite, nous introduirons les entrelaceurs convolutionnels qui ne sont pas plus complexes que les entrelaceurs blocs, mais qui ont le désavantage d'induire des délais à la transmission de l'information.

6.1 Les entrelaceurs blocs

6.1.1 Principe

Les entrelaceurs blocs sont des systèmes très simples. Ils font partie de ce qu'on appelle les entrelaceurs déterministes, à savoir que nous connaissons à l'avance la position d'un bit après entrelacement. Quelle que soit la longueur de bloc utilisée, pour générer un entrelaceur bloc, il suffit d'écrire les bits d'entrée dans une matrice, par exemple ligne par ligne. La lecture des bits entrelacés se fait alors colonne par colonne. Il existe de nombreuses façons de générer un tel entrelaceur. Par exemple, nous pourrions écrire la matrice d'entrée ligne par ligne de droite à gauche et lire colonne par colonne de haut en bas. Toutefois, les performances de ces entrelaceurs ne dépendent pas de la procédure de lecture. Nous avons utilisé la méthode classique d'écriture ligne par ligne de gauche à

droite et d'écriture colonne par colonne de haut en bas. Un entrelaceur bloc est illustré à la figure 6.1. À la figure 6.2, nous avons également illustré le processus d'entrelacement bloc. Avec la nomenclature que nous avons prise, deux bits consécutifs à l'entrée seront séparés du nombre de lignes. Ainsi, à la figure 6.2, nous voyons que les bits 1 et 2 sont séparés de 4 bits, ce qui correspond aux nombres de lignes de la matrice d'entrelacement. Il serait donc judicieux de prendre un nombre important de lignes pour la matrice d'entrelacement. Toutefois, il faut prendre en compte le fait que le nombre de colonnes doit aussi être important, sinon, l'entrelacement ne serait pas judicieux.

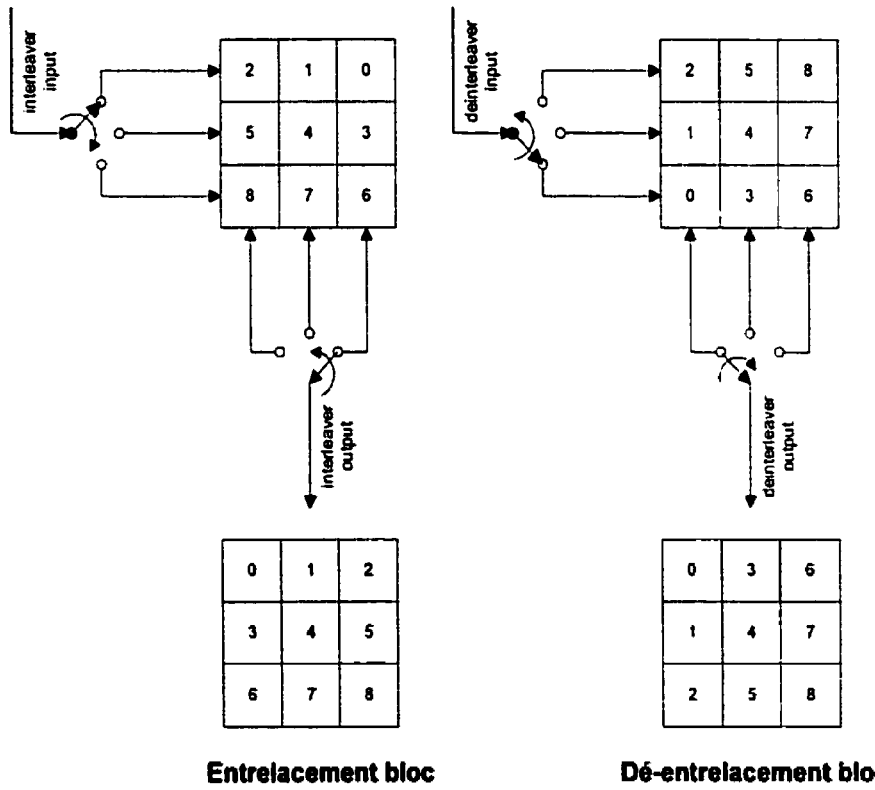


Figure 6.1 – Entrelacement bloc

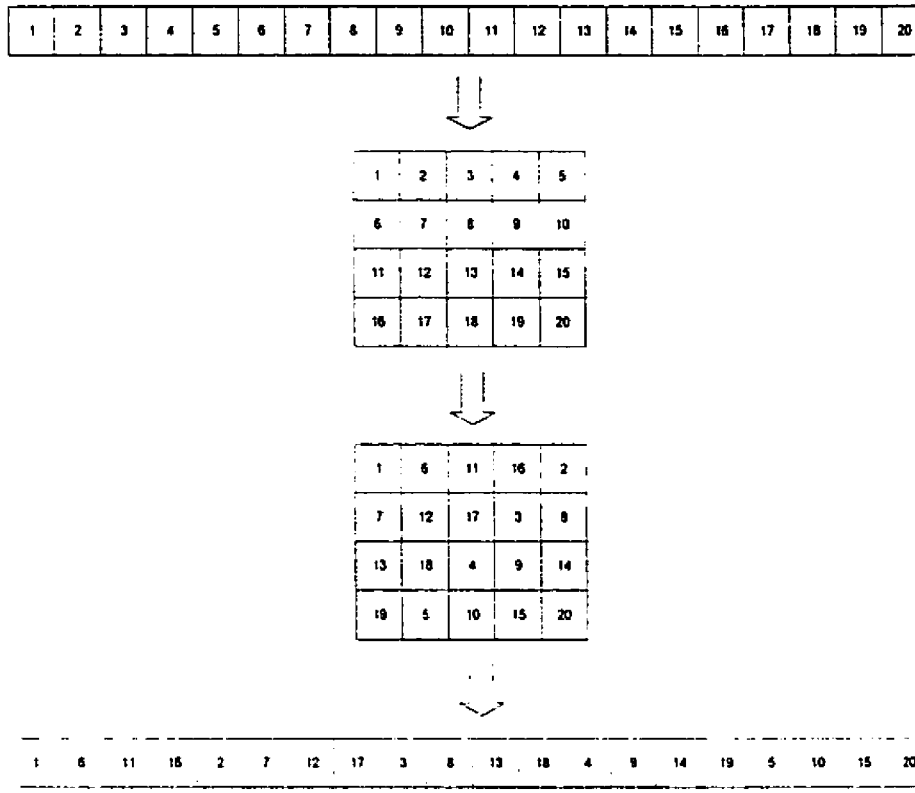


Figure 6.2 – Processus d'entrelacement bloc

6.1.2 Résultats

Nous avons effectué des simulations pour évaluer la qualité de l'entrelaceur bloc. Nous avons simulé un codeur turbo de taux global 1/3, de longueur de contrainte K = 3, dans un canal AWGN. Nous avons effectué nos simulations pour des longueurs d'entrelaceurs entre 196 bits et 900 bits. Nous allons, dans ce paragraphe, présenter nos résultats.

6.1.2.1 Entrelaceurs de longueur 196 bits

Il s'agit ici d'un nombre peu élevé de bits. Comme nous pouvons le constater à la figure 6.4, le facteur d'étalement d'un tel entrelaceur semble bon. Les bits après entrelacement sont bien répartis. Il en ressort une bonne performance de ces entrelaceurs. Il est à noter

que l'entrelaceur bloc est très simple. Cette information sera à prendre en compte lors de la comparaison des entrelaceurs.

Comme nous l'avons déjà noté, le gain entre les différentes itérations est excellent entre la première et la deuxième itération, puis entre la deuxième et la troisième. Toutefois, ce gain est très restreint quand on passe aux itérations suivantes. Nous observons aussi ici une certaine saturation des Codes Turbo. Ce type d'entrelacement tend donc, comme tous ceux que nous avons vus jusqu'à maintenant à ne plus pouvoir apporter de gain au décodage. Globalement, ce type d'entrelaceur apporte une bonne performance pour de petites tailles de blocs. Nous allons maintenant voir s'il en est de même pour de plus grandes tailles de blocs.

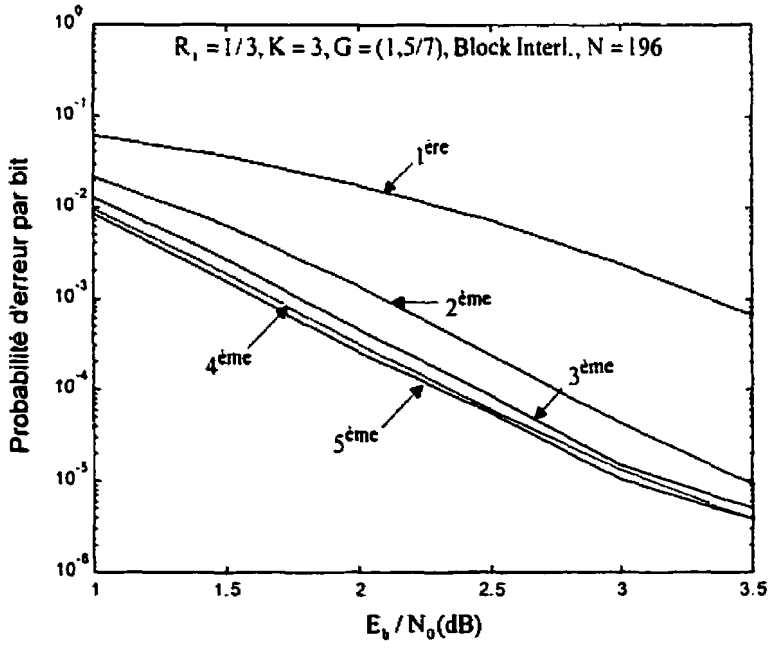


Figure 6.3 – BER, $R_t = 1/3$, K = 3, entrelaceur bloc, canal AWGN, N = 196

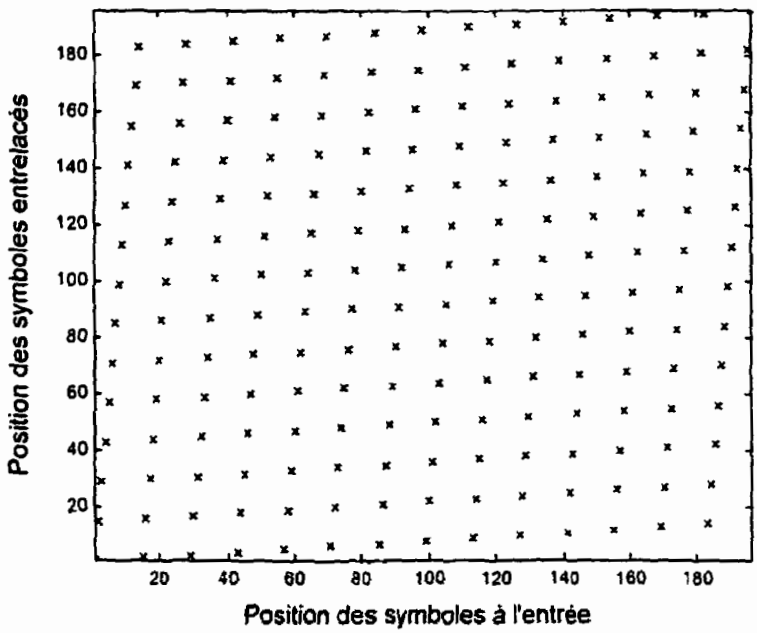


Figure 6.4 – Facteur d'étalement, entrelaceur bloc, N = 196

6.1.2.2 Entrelaceurs de longueur 400 bits

Nous nous sommes donc maintenant intéressés à un nombre de bits plus élevés, à savoir 400. En observant le facteur d'étalement d'un entrelaceur bloc de 400 bits, nous pouvons conclure qu'il est assez bon. Maintenant, nous remarquons que ce facteur est uniforme et que la distance après entrelacement est constante. Il devrait en résulter une très bonne performance. Il n'en est malheureusement pas le cas. En effet, si nous regardons la figure 6.5, nous notons que les performances sont bonnes mais semblent saturer rapidement. Ce type d'entrelacement semble donner de bonnes performances pour un petit nombre d'itérations. Nous voyons que dès la deuxième itération, nous obtenons des performances semblables à la quatrième ou cinquième itération. Ce genre d'entrelaceur n'est donc pas intéressant pour des simulations à de nombreuses itérations. Il semblerait donc que l'entrelaceur bloc soit excellent pour de petits blocs et ne soit plus très intéressant lorsque

nous augmentons la taille des blocs. Nous verrons si cette affirmation est vraie dans le prochain paragraphe lors de la présentation des résultats pour des blocs de 900 bits.

En ce qui concerne le gain entre les différentes itérations, il n'est pas très bon. En effet, dès la deuxième itération, nous obtenons des performances similaires aux itérations supérieures pour des rapports signal sur bruit élevés. Toutefois, pour des rapports où le bruit est important, utiliser un plus grand nombre d'itérations semble intéressant. Le gain à 10^{-3} entre la deuxième et la troisième itération est d'environ 0.3 dB, alors qu'il est d'environ 0.5 dB à 10^{-4}. Globalement, ce type d'entrelaceur est bon pour des tailles de bloc de 400 bits, mais non pour un petit nombre d'itérations, étant donnée la saturation rapide des Codes Turbo. Nous allons maintenant nous intéresser à des tailles de bloc de 900 bits.

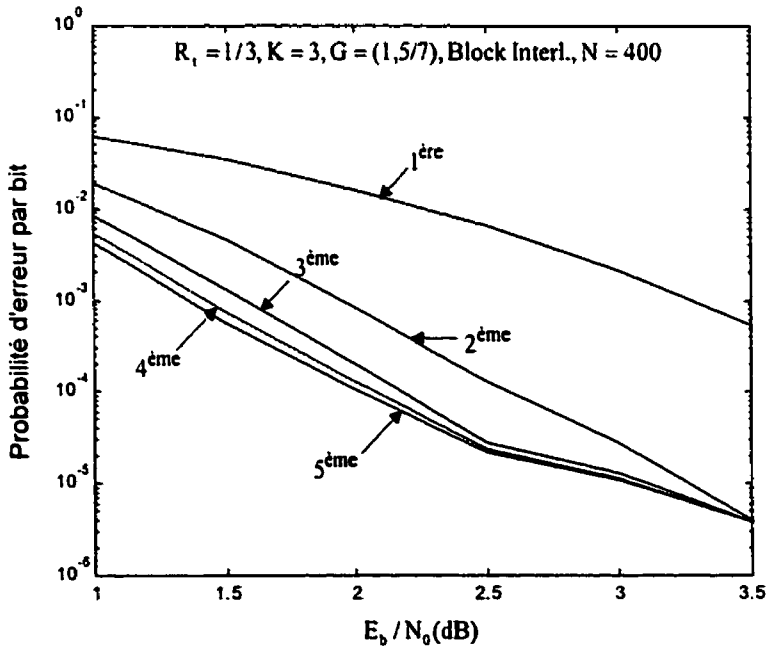


Figure 6.5 – BER, $R_t = 1/3$, $K = 3$, entrelaceur bloc, canal AWGN, $N = 400$

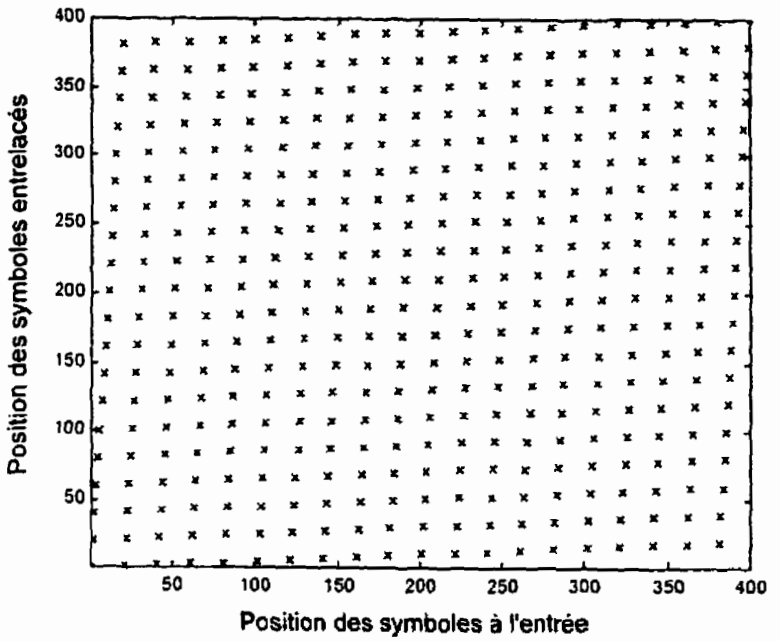


Figure 6.6 – Facteur d'étalement, entrelaceur bloc, N = 400

6.1.2.2 Entrelaceurs de longueur 900 bits

Pour une telle taille de blocs, en observant le facteur d'étalement à la figure 6.8, il semble que plus le nombre de bits augmente, moins le facteur d'étalement semble bon. Alors que pour des tailles de 196 et 400 bits, ce paramètre semblait bien répartir les bits après entrelacement, il ne semble plus être de même pour 900 bits. L'uniformité de la répartition après entrelacement qui caractérise cet entrelaceur semble donc jouer en sa défaveur pour de grandes tailles de bloc. En effet, plus on augmente la taille, moins cet entrelaceur semble être capable de décorréler l'information. À 900 bits, la place pour la répartition des bits est plus importante que pour de plus petites tailles de bloc. Un entrelaceur déterministe qui prédétermine les places des bits après entrelacement n'utilisera donc pas de façon adéquate cet espace pour des fins de décorrélation.

Ceci se reflète sur les performances à la figure 6.7. Nous observons qu'il n'est pas nécessaire d'utiliser un grand nombre d'itérations étant donné que les itérations trois, quatre et cinq donnent les mêmes performances. Nous remarquons également que, comme pour 400 bits, le gain entre la deuxième et la troisième itération est mince, et il est encore moins important que pour le cas de 400 bits. Ce type d'entrelaceur sature donc encore plus rapidement pour de grandes tailles de blocs, ce qui peut s'avérer inintéressant dans le cas de transmissions par satellites. Toutefois, nous avons vu que pour de petites tailles de bloc, les performances sont intéressantes pour une complexité moindre. Les entrelaceurs déterministes semblent donc bons pour de petites tailles de bloc. Nous allons maintenant tenter de confirmer ceci en étudiant un deuxième type d'entrelaceur déterministe, à savoir les entrelaceurs convolutionnels.

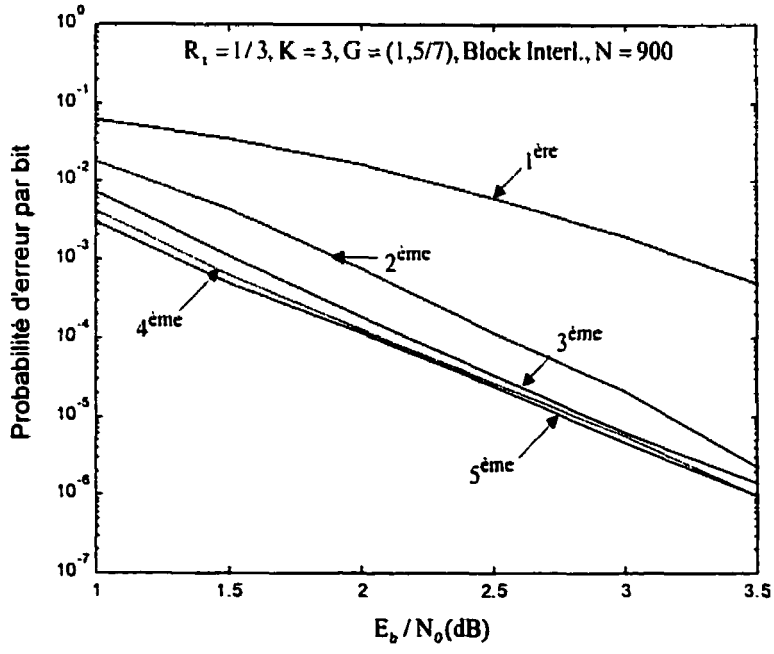


Figure 6.7 – BER, $R_t = 1/3$, $K = 3$, entrelaceur bloc, canal AWGN, $N = 900$

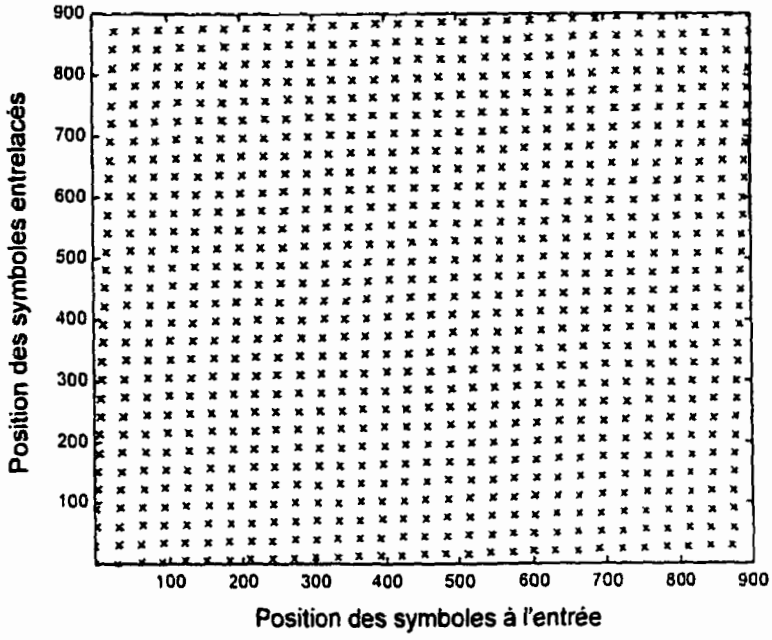


Figure 6.8 – Facteur d'étalement, entrelaceur bloc, N = 900

6.2 Les entrelaceurs convolutionnels

6.2.1 Principe

Les entrelaceurs convolutionnels, avec les blocs, font partie des moyens les plus fréquemment utilisés dans les systèmes de transmissions par canal. Les entrelaceurs convolutionnels sont aussi appelés "cross interleave".

Un tel entrelaceur est dessiné à la figure 1. Le circuit est caractérisé par son indice m et le nombre de lignes de délai D. Chaque bloc intitulé D sur la figure correspond à un délai, qui peut aussi bien être égal à 1 bit ou plus. Comme indiqué sur la figure 6.9, un commutateur permet de répartir les bits d'entrée sur m différentes lignes. La caractéristique de cet entrelaceur est qu'à la $i^{ème}$ ligne, le bit est introduit avec un délai de i x D bits. Ainsi, à la troisième ligne, si nous considérons des délais de 1 bit, chaque bit

aura un décalage de 3 le long de la ligne. À la sortie, un autre commutateur permet la lecture des bits. Nous observons que le premier bit sera lu dès le départ. Une fois ce bit lu, l'interrupteur passe à la seconde ligne. Normalement, le bit de sortie devrait être le deuxième transmis. Néanmoins, il faut prendre en compte le délai. Ce dernier implique qu'aucun bit ne sera lu à cet instant. Étant donné qu'il y a des délais sur chacune des lignes, le deuxième bit lu à la sortie sera le deuxième bit qui a été introduit sur la première ligne. Il s'agit donc du $(m+1)^{ème}$ bit car il y a m lignes. Ce n'est qu'à la deuxième itération de l'interrupteur qu'un bit de la deuxième ligne est lu. Ceci peut se généraliser: un bit de la $i^{ème}$ ligne sera lu, pour la première fois, au $i^{ème}$ passage de l'interrupteur.

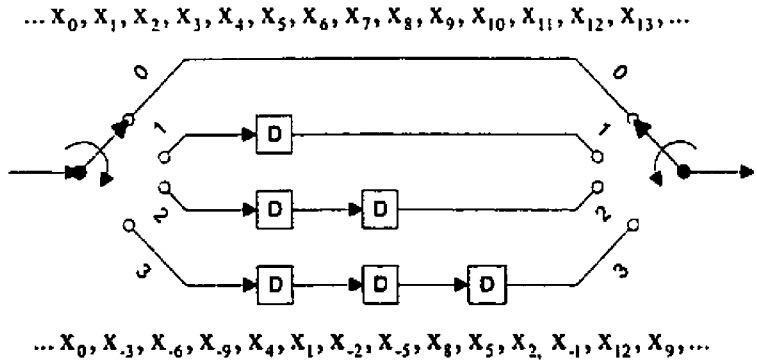


Figure 6.9 - Processus d'entrelacement convolutionnel (m = 4, D = 1 bit)

Notons un délai par le symbole *. L'entrelacement convolutionnel peut alors être vu comme une matrice, dans la laquelle les bits d'entrée sont introduits ligne par ligne. Ceci est représenté à la figure 6.10. À titre de représentation, nous avons utilisé un bloc de 20 bits entrelacés avec les paramètres m = 4 et D = 1.

1	5	9	13	17	*	*	*
*	2	6	10	14	18	*	*
*	*	3	7	11	15	19	*
*	*	*	4	8	12	16	20

Figure 6.10 - Matrice de l'entrelacement convolutionnel m = 4, D = 1

La séquence de sortie est alors obtenue en lisant les bits colonne par colonne:

1	*	*	*	5	2	*	*	9	6	3	*	13	10	7	4
17	14	11	8	*	18	15	12	*	*	19	16	*	*	*	20

Figure 6.11 - Séquence de sortie de l'entrelacement convolutionnel m = 4, D = 1

Nous constatons, à l'aide de la figure 6.10, que le mélange des bits se fait de façon hélicoïdale si on ne prend pas en compte les délais. En ce qui concerne, la séparation des bits après entrelacement, la distance minimale entre deux bits consécutifs après entrelacement est au moins m, étant donné qu'il y a m lignes. Ceci est en général valable pour tout D. En effet, illustrons ceci en comparant les séquences de sortie pour le même entrelaceur de la figure 1, mais avec D = 1 et D = 2. La figure 6.11 illustre la sortie pour D = 1. La matrice d'entrelacement pour D = 2 est indiquée à la figure 6.12. Quant à la séquence de sortie pour D = 2, elle est illustrée à la figure 6.13.

1	5	9	13	17	*	*	*	*	*	*
*	*	2	6	10	14	18	*	*	*	*
*	*	*	*	3	7	11	15	19	*	*
*	*	*	*	*	*	4	8	12	16	20

Figure 6.12 - Matrice de l'entrelacement convolutionnel m = 4, D = 2

1	*	*	*	5	*	*	*	9	2	*
*	13	6	*	*	17	10	3	*	*	14
7	*	*	18	11	4	*	*	15	8	*
*	19	12	*	*	*	16	*	*	*	20

Figure 6.13 - Séquence de sortie de l'entrelacement convolutionnel m = 4, D = 2

Grâce à la figure 6.13, nous constatons que la séparation entre les bits pour D = 2 est meilleure que pour D = 1, c'est-à-dire 9 au lieu de 5. En effet, la séparation après entrelacement entre le bit 11 et le bit 12 est bien 9 par exemple. Nous avons donc gagné

en terme de séparation. Ceci se généralise évidemment: plus D est grand, meilleur est le spectre des séparations. Toutefois, il faut bien comprendre que plus D augmente, plus longue sera la durée de l'encodage. En terme de délais, nous en avons introduit deux fois plus pour le cas D = 2. En effet, pour D = 1, nous pouvons calculer 12 délais introduits, alors que pour D = 2, il en faut 24 pour générer la matrice. Ce gain en terme de séparation s'accompagne donc d'un délai plus important. Il faudra donc bien prendre en compte ces paramètres lors du choix d'un entrelaceur convolutionnel.

6.2.2 Résultats

Nous avons effectué des simulations pour évaluer la qualité de l'entrelaceur convolutionnel. Nous avons simulé un codeur turbo de taux global 1/3, de longueur de contrainte 3, dans un canal AWGN. Nous avons effectué nos simulations pour des longueurs d'entrelaceurs entre 196 bits et 900 bits. Nous allons, dans ce paragraphe, présenter nos résultats.

6.2.2.1 Entrelaceurs de longueur 196 bits

6.2.2.1.1 Variation des paramètres m et D

Comme nous avons vu au paragraphe précédent, un entrelaceur convolutionnel est caractérisé par deux paramètres, à savoir m le nombre de lignes de l'entrelaceur et D le nombre de délais. Il s'ensuit donc que les performances de cet entrelaceur vont varier en fonction de ces paramètres. Une méthode intéressante à utiliser pour l'étude des performances de cet entrelaceur pour les Codes Turbo est donc de faire varier ces deux paramètres indépendamment l'un de l'autre. Notre étude commence donc avec un graphique illustrant l'évolution des performances de l'entrelaceur convolutionnel en variant m et D. À la figure 6.14 se trouve un graphique de cette évolution. Nous avons

pris les résultats que nous avons obtenus à 2.5 dB et pour la quatrième itération. Nous avons fait varier le nombre de lignes de 4 à 30 et le nombre de délais par bloc D de 1 à 4.

Ce graphe va nous permettre de déterminer la meilleure combinaison possible pour un tel entrelaceur. Nous observons que plus D augmente, meilleures sont les performances en général. Toutefois, nous notons également que plus nous augmentons le nombre de lignes m, moins cela s'avère intéressant. En effet, par exemple, pour m = 25, nous observons les mêmes performances que ce soit pour D =2, D = 3 ou D = 4. Donc, pour un tel nombre de lignes, il ne s'avère pas judicieux d'augmenter le nombre de délais étant donné que ceci ajoute de la complexité sans améliorer les performances.

La variation du nombre de lignes donne une courbe en "U" et nous pouvions nous attendre à ceci. En effet, le nombre de bits par bloc que nous avons pris est 196. Pour un petit nombre de lignes, nous avons donc un grand nombre de colonnes et l'entrelacement n'est pas efficace. De même, pour un grand nombre de lignes, nous avons un petit nombre de colonnes, nous donnant ainsi un aspect symétrique à cet entrelacement. Les meilleures performances sont obtenues pour m = 17. De surplus, nous observons que la meilleure combinaison est (17,4).

Néanmoins, nous pouvons nous demander si le gain en performances justifie d'aller jusqu'à D = 4. Autant le passage entre D = 1 et D = 2 est intéressant, autant il n'en est peut-être pas de même pour les autres. Ce passage entre D = 1 et D = 2 nous donne une forte amélioration dans la performance vu que nous passons de 5.10^{-4} à 9.10^{-5}. Par la suite, nous ne gagnons pas autant. Passer de D = 2 à D = 3 améliore la performance de 9.10^{-5} à 8.10^{-5}, gain très minime par rapport au précédent. Enfin, nous ne notons aucun gain n'est observé entre D = 3 et D = 4, il s'agit même d'une perte de performances. Ceci nous amène donc à conclure qu'à partir d'un certain D, plus aucun gain n'est observé. Dans ce cas, il s'agit de D = 3. Il est donc judicieux de ne pas augmenter la complexité inadéquatement.

Nous sommes certains maintenant que la meilleure combinaison est (17,3) ou (17,4). Toutefois, si à 2.5 dB, nous recherchons une BER de 10^{-4}, nous avons quatre possibilités à l'aide de ce graphique. Nous pouvons utiliser soit (17,1), soit (9,2), soit (7,3), ou encore (6,4). Dans ce cas, il serait plus judicieux de prendre le premier cas à cause du délai induit restreint. Toutefois, il se peut que nous ne puissions remplir nos exigences avec $D=1$. Ceci serait le cas si nous recherchions une BER meilleure que 10^{-4}. Dans ce cas, nous nous rabattrions sur D = 2. Nous voyons donc que ce genre de graphe peut s'avérer très utile dans le choix de notre combinaison de lignes et de délais. Il suffit de tracer le bon graphe au bon rapport signal sur bruit et à la bonne itération. Finalement, nous aurions pu effectuer des simulations impliquant des délais supérieurs à 4. Toutefois, nous nous sommes arrêtés à 4 car le but de ce paragraphe est de montrer l'évolution des performances. De surplus, nous avons vu qu'à partir de D = 3, nous ne gagnions plus grand chose en performances.

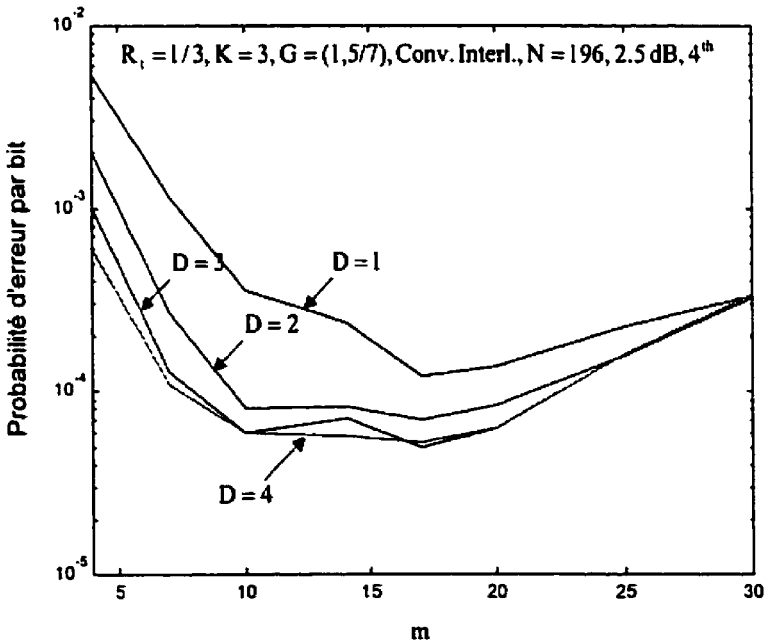


Figure 6.14 – BER de l'entrelaceur convolutionnel en fonction de m et D pour N = 196

6.2.2.1.2 Évaluation des entrelaceurs convolutionnels de longueur 196 bits

Nous venons de voir qu'il existe une combinaison idéale pour ce type d'entrelaceur. Nous allons maintenant tenter de justifier ce comportement. Aux figures 6.15, 6.16, 6.17 et 6.18, nous avons tracé les facteurs d'étalement de quatre entrelaceurs convolutionnels de 196 bits. Progressivement, nous avons augmenté le nombre de lignes et le nombre de délais.

La première figure illustre un cas extrême, à savoir (10,1). Nous avons pris ce cas pour montrer que les bits entrelacés ne sont pas bien répartis, ce qui est le cas. Le passage à la prochaine figure illustre le meilleur cas pour D = 1, à savoir 17 lignes. Nous notons tout de suite une meilleure répartition, mais nous observons encore des zones inutilisées. Ceci implique donc qu'il y a sujet à amélioration. Nous avons déjà admis que la meilleure combinaison pour D = 1 est m = 17. Ainsi, si nous voulons améliorer les performances, il va falloir passer à un D supérieur. Nous avons décidé de prendre D = 4 pour illustrer la différence.

Pour m = 17 et D = 4, à la figure 6.17, nous notons une bonne répartition des bits. Ceci confirme le bon comportement de cet entrelaceur pour cette combinaison. Enfin, nous avons illustré à la figure 6.18 l'effet d'un m trop grand. Utiliser un m trop élevé entraîne une répartition trop dense des bits et implique une mauvaise performance.

Finalement, à la figure 6.19, nous avons tracé la courbe des performances des Codes Turbo de l'entrelaceur (17,4) qui nous est apparu comme le meilleur. Ce type d'entrelaceur a un comportement très personnel. En effet, nous notons un bon gain entre la première et la deuxième itération ainsi qu'entre les deux itérations subséquentes. Toutefois, à partir de l'itération numéro trois, nous obtenons les mêmes performances. Il est donc évident qu'il est inutile d'aller au-delà de la troisième itération. Globalement, ce type d'entrelaceur ne possède pas de bonnes performances, mais pour s'en persuader, il

faudra le faire en effectuant la comparaison avec les autres entrelaceurs, ce qui fera l'objet de notre dernier chapitre.

Maintenant que nous avons effectué l'étude pour des blocs de 196 bits, nous allons nous intéresser à des longueurs de blocs plus importantes pour voir si ce type d'entrelaceur possède un meilleur comportement.

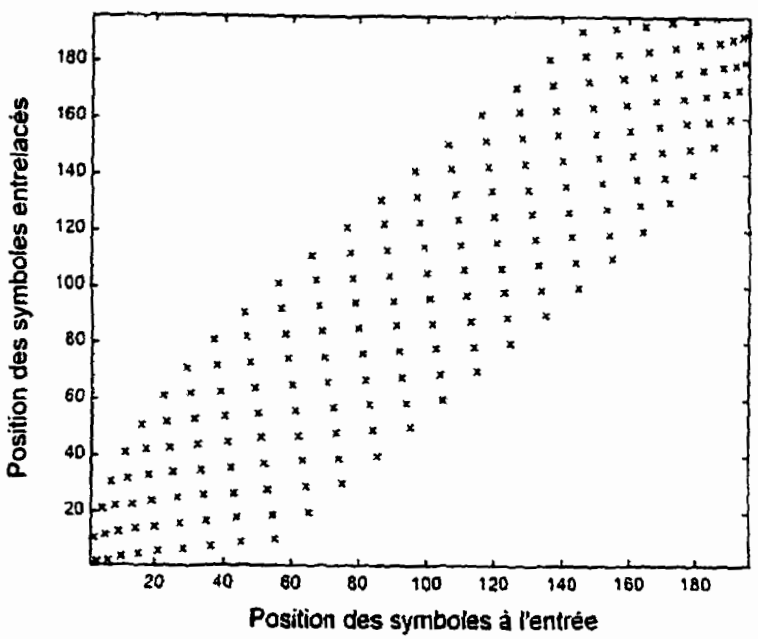


Figure 6.15 – Facteur d'étalement, entrelaceur convolutionnel (10,1), N = 196

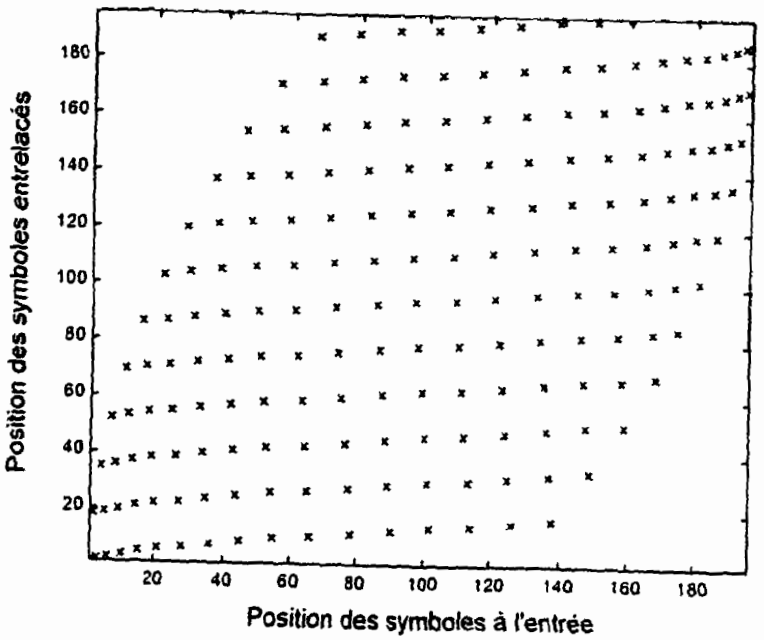


Figure 6.16 – Facteur d'étalement, entrelaceur convolutionnel (17,1), N = 196

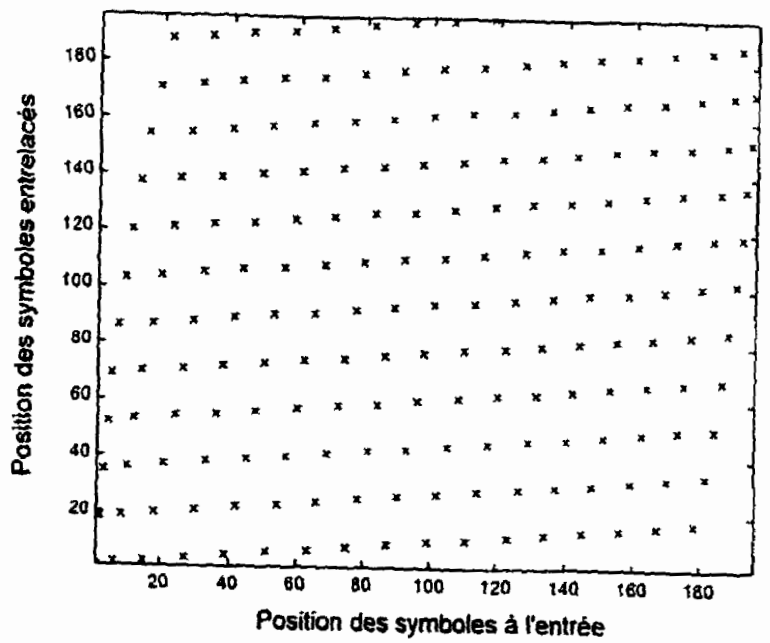


Figure 6.17 – Facteur d'étalement, entrelaceur convolutionnel (17,4), N = 196

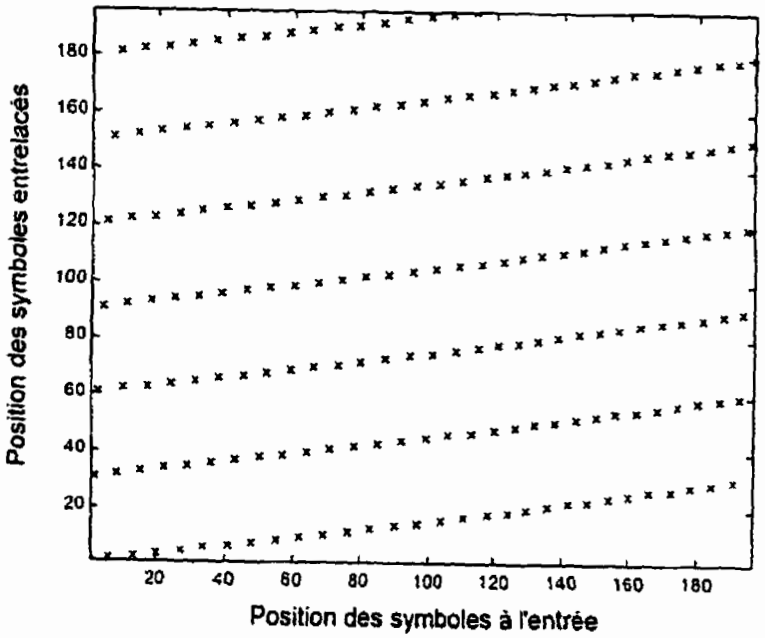


Figure 6.18 – Facteur d'étalement, entrelaceur convolutionnel (30,4), N = 196

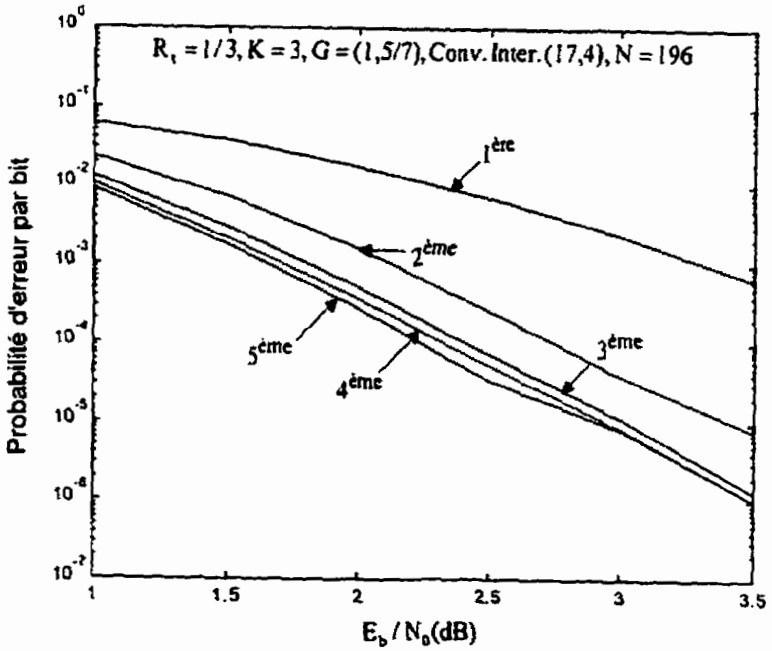


Figure 6.19 – BER, $R_t = 1/3$, K = 3, ent. convolutionnel (17,4), canal AWGN, N = 196

6.2.2.2 Entrelaceurs de longueur 400 bits

6.2.2.2.1 Variation des paramètres m et D

Nous avons effectué la même étude que pour des blocs de 196 bits. Nous avons tracé le même type de courbes et nous en sommes arrivés aux même conclusions. En effet, nous observons toujours le comportement en "U". Toutefois, cette fois, la meilleure combinaison est différente. En effet, à la figure 6.20, nous observons que le nombre de lignes idéal est 30. Ce nombre a donc augmenté par rapport aux blocs de 196 bits. Ceci est normal car nous considérons maintenant des tailles de blocs supérieures.

L'influence des délais ne change pourtant pas. Effectivement, plus nous augmentons ce dernier, meilleures sont les performances. Nous notons même cette fois un gain entre $D=1$ et $D=4$. Ceci est à mettre au compte que comme nous utilisons des blocs de 400 bits, la matrice d'entrelacement est plus grande et offre plus de place à l'entrelacement des bits. Nous observons toutefois que le gain entre D = 3 et D = 4 est minime.

Ainsi, globalement, le comportement de cet entrelaceur est le même pour des blocs de 196 bits. La meilleure combinaison, cette fois, est (30,4).

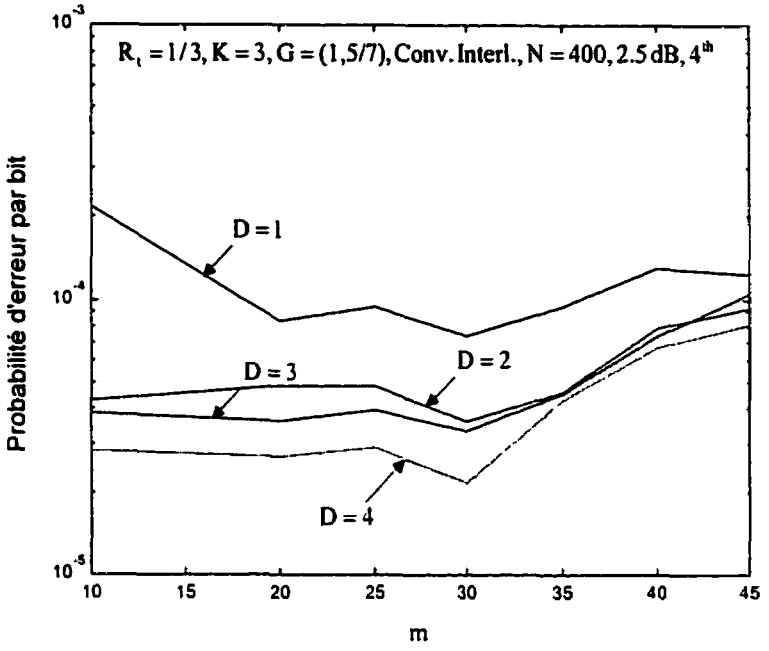


Figure 6.20 – BER de l'entrelaceur convolutionnel en fonction de m et D pour N = 400

6.2.2.2.2 Évaluation des entrelaceurs convolutionnels de longueur 400 bits

Comme pour des blocs de 196 bits, nous avons tracé les facteurs d'étalement de différentes combinaisons aux figures 6.21, 6.22, 6.23 et 6.24. Nous avons suivi le même raisonnement. Tout d'abord, nous nous sommes intéressés à la combinaison (10,1) qui nous donne une répartition très dense laissant ainsi des zones non occupées. Ensuite, nous avons utilisé la combinaison (30,1) qui est la meilleure pour D = 1. Nous observons exactement le même comportement que pour des blocs de 196 bits. En effet, il existe bel et bien une amélioration dans la répartition des bits, mais il reste toutefois des zones non occupées, laissant ainsi la possibilité d'une amélioration.

Nous obtenons cette amélioration en augmentant le délai. Nous avons alors utilisé la combinaison (30,4) qui est censée être la meilleure des combinaisons que nous avons utilisées. Cette fois, les bits sont bien répartis et donnent ainsi de bonnes performances

que nous avons tracées à la figure 6.25. Ces performances sont exactement du même type que pour N = 196. Les mêmes conclusions peuvent être dressées, à savoir que la meilleure itération est la troisième, et que les performances ne semblent pas excellentes. Enfin, nous avons augmenté le nombre de ligne jusqu'à 45 et nous avons observé la mauvaise répartition des bits.

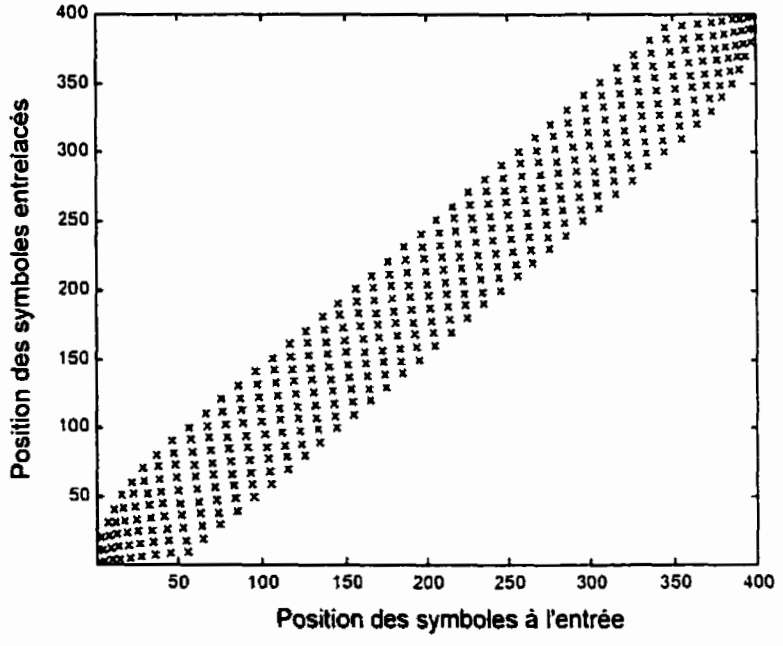


Figure 6.21 – Facteur d'étalement, entrelaceur convolutionnel (10,1), N = 400

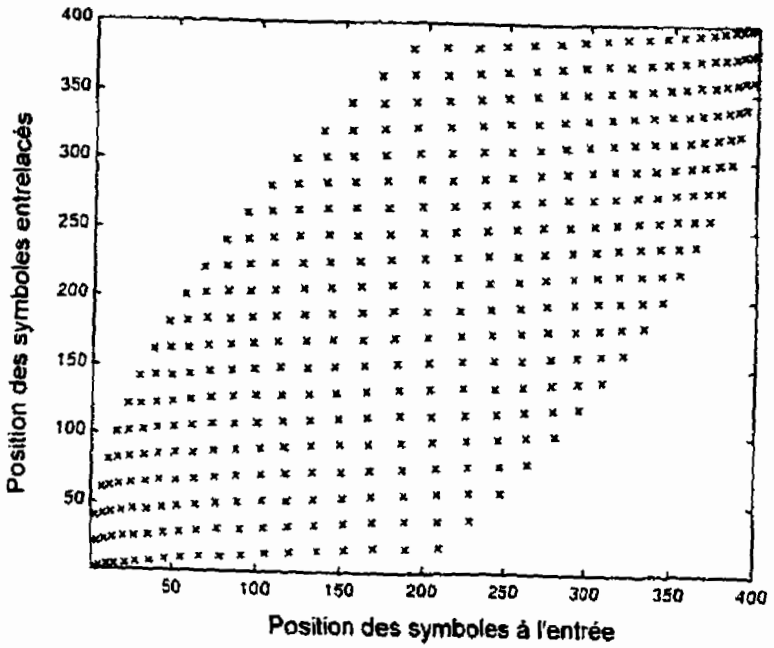


Figure 6.22 – Facteur d'étalement, entrelaceur convolutionnel (30,1), N = 400

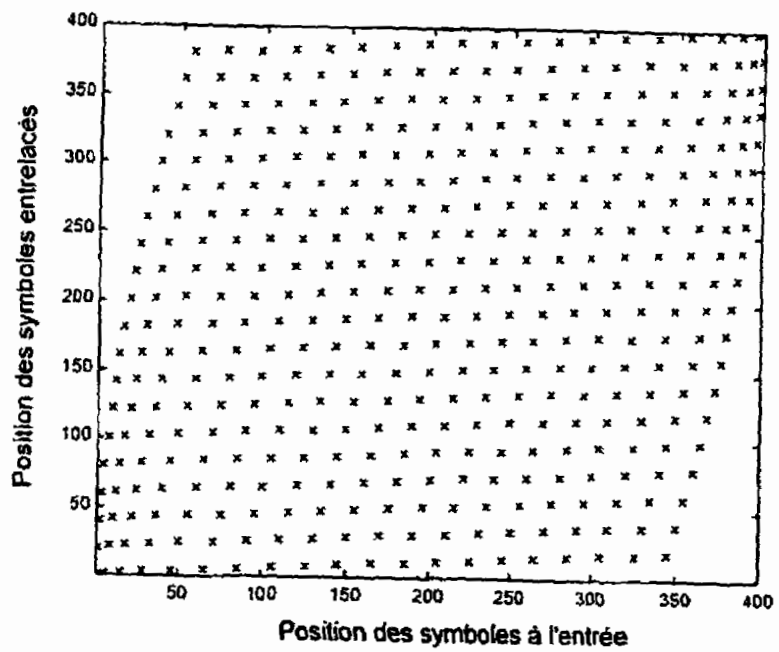


Figure 6.23 – Facteur d'étalement, entrelaceur convolutionnel (30,4), N = 400

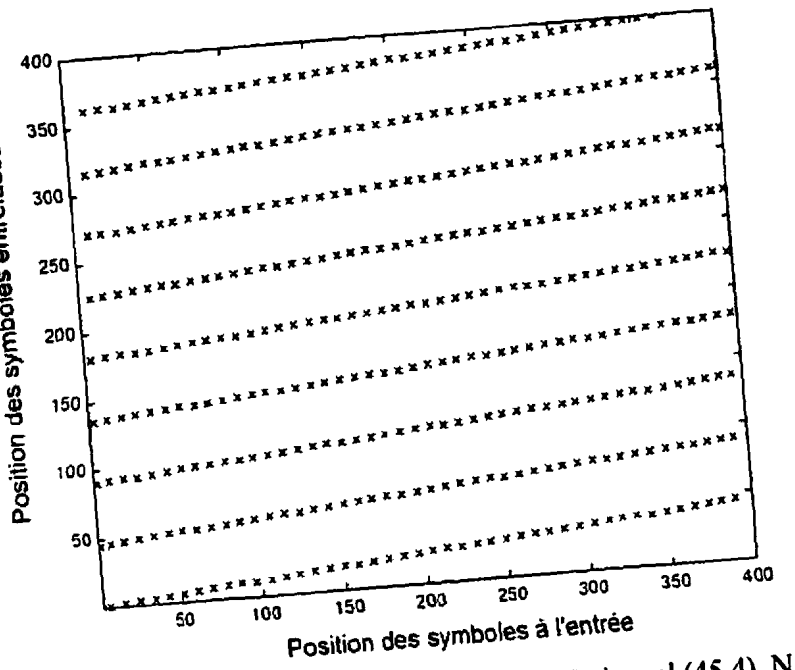


Figure 6.24 – Facteur d'étalement, entrelaceur convolutionnel (45,4), N = 400

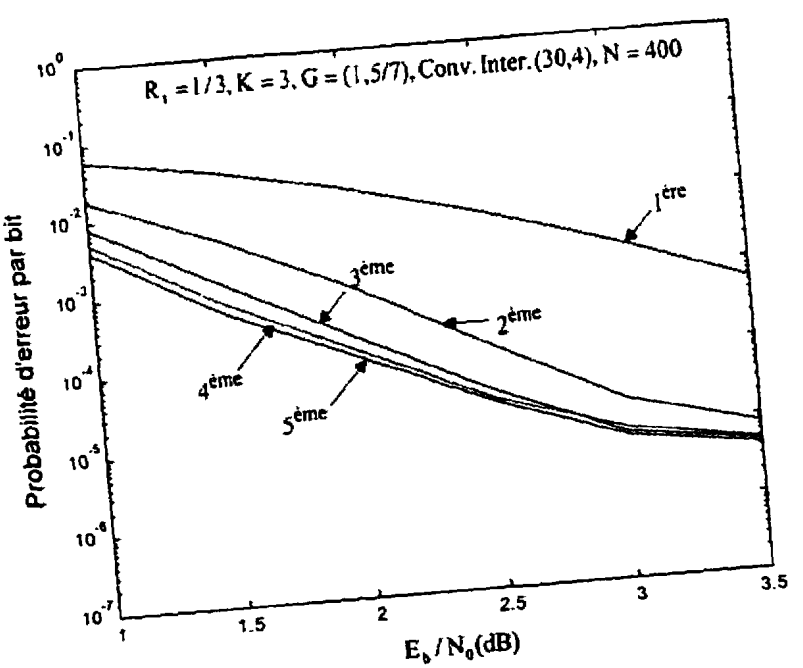


Figure 6.25 – BER, R_t = 1/3, K = 3, ent. convolutionnel (30,4), canal AWGN, N = 400

6.2.2.3 Entrelaceurs de longueur 900 bits

Nous avons effectué les mêmes simulations pour N = 900. Étant donné que nous arrivons aux mêmes conclusions qu'aux paragraphes précédents, nous avons juste tracé les courbes dans ce paragraphe avec une succincte explication.

6.2.2.3.1 Variation des paramètres m et D

Dans ce cas, il semble que le nombre de lignes optimal est 35. Nous notons aussi qu'au-delà d'un délai de 2, aucun gain n'est observé. Il semblerait donc que plus nous augmentons la taille des blocs, moins le délai a d'influence.

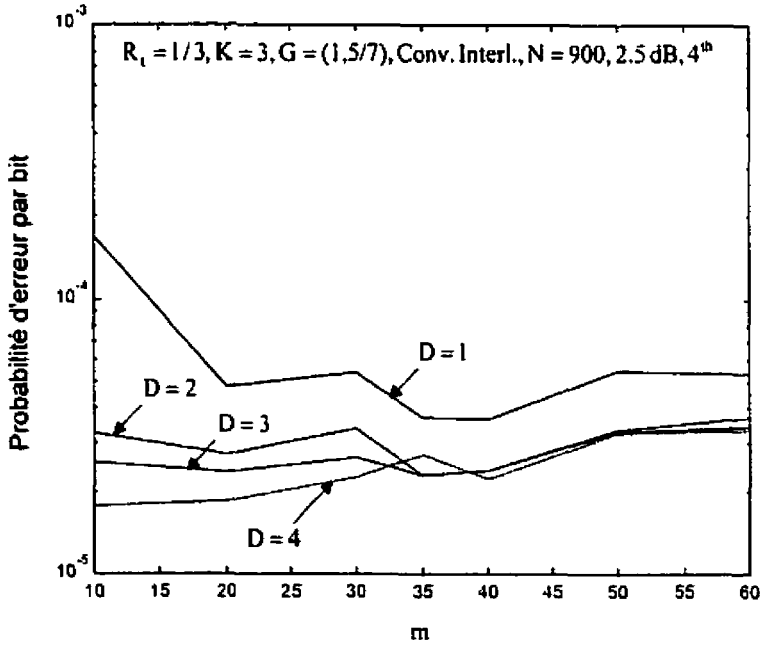


Figure 6.26 – BER de l'entrelaceur convolutionnel en fonction de m et D pour N = 900

6.2.2.3.2 Évaluation des entrelaceurs convolutionnels de longueur 900 bits

Les facteurs d'étalement de quatre entrelaceurs convolutionnels de 900 bits ont été tracés aux figures 6.27, 6.28, 6.29, et 6.30. Enfin, les performances ont été évaluées à la figure 6.31.

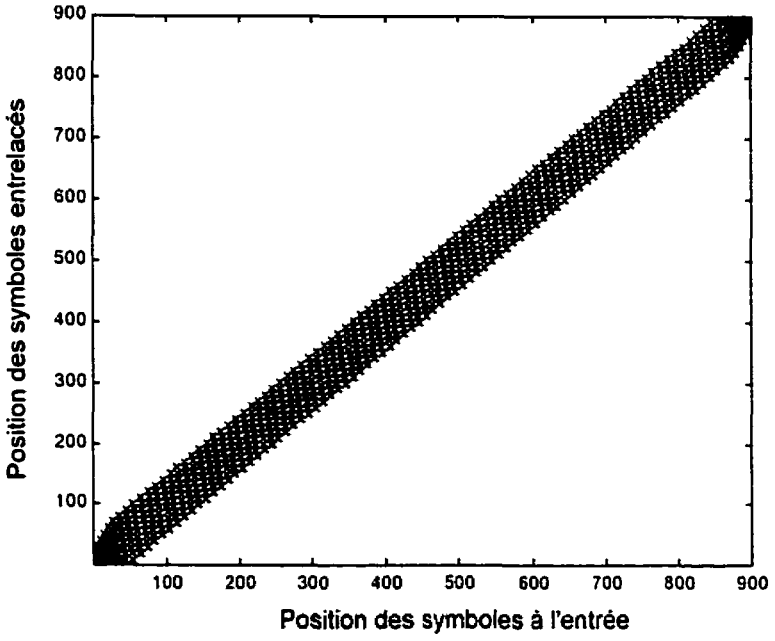


Figure 6.27 – Facteur d'étalement, entrelaceur convolutionnel (10,1), N = 900

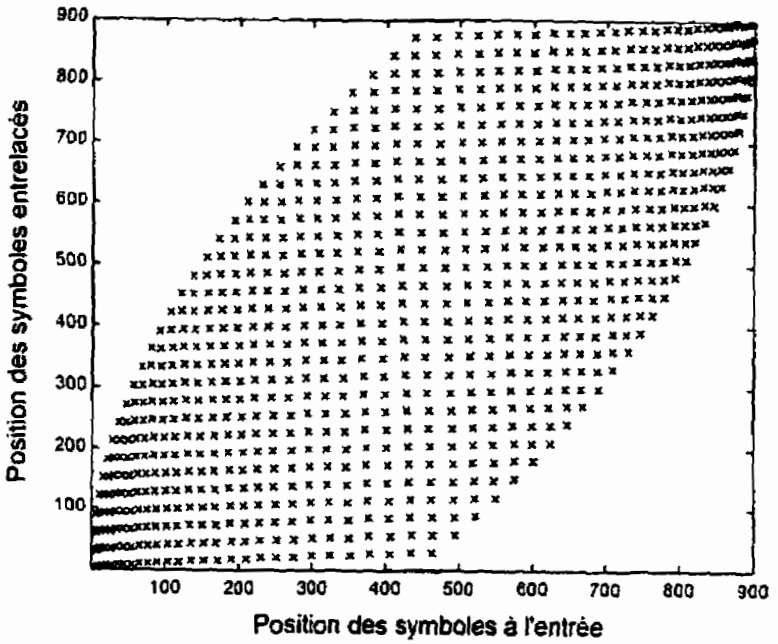


Figure 6.28 – Facteur d'étalement, entrelaceur convolutionnel (35,1), N = 900

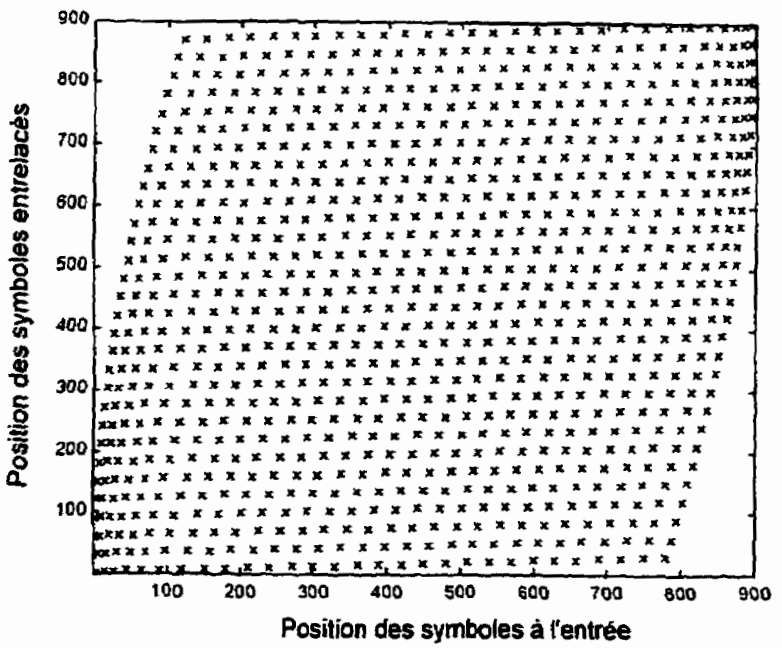


Figure 6.29 – Facteur d'étalement, entrelaceur convolutionnel (35,4), N = 900

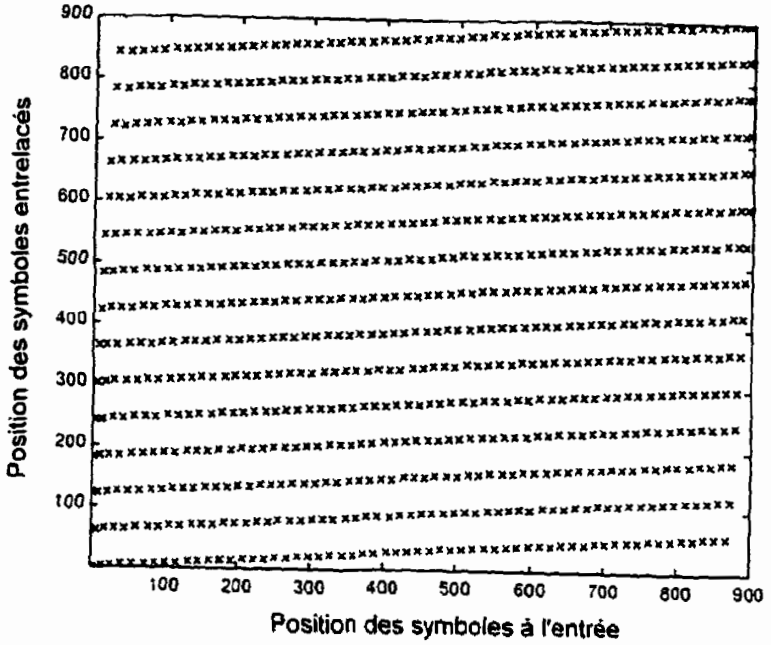


Figure 6.30 – Facteur d'étalement, entrelaceur convolutionnel (60,4), N = 900

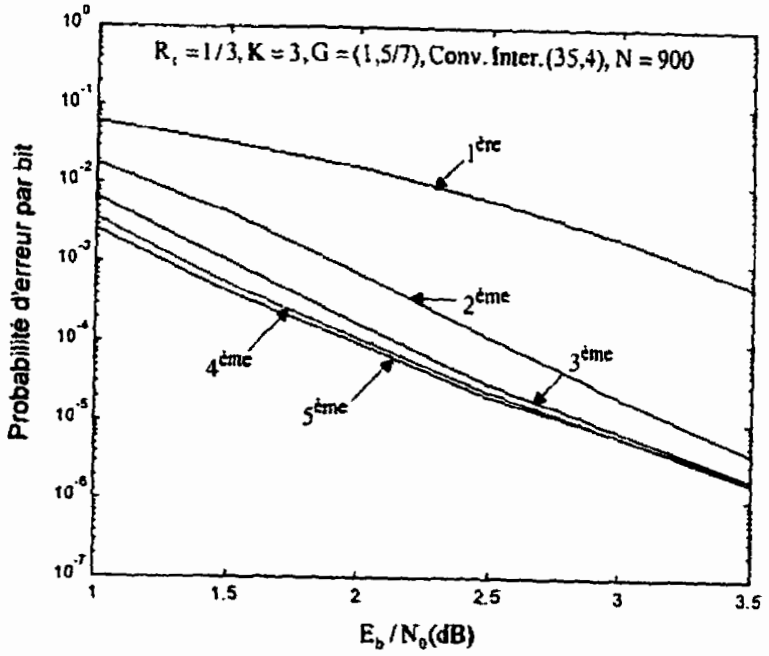


Figure 6.31 – BER, $R_t = 1/3$, $K = 3$, ent. convolutionnel (35,4), canal AWGN, $N = 900$

6.3 Conclusion

Dans ce chapitre, nous nous sommes donc intéressés à des entrelaceurs plus classiques, à savoir les entrelaceurs déterministes. Ces derniers sont caractérisés par la connaissance de la position des bits par avance. Nous avons vu, dans un premier temps, les entrelaceurs blocs qui sont les plus classiques et les plus simples. Toutefois, bien que ces derniers possèdent de bonnes performances pour de petites tailles de bloc, ils se détériorent rapidement à mesure que nous augmentons N.

Par la suite, nous avons étudié les entrelaceurs convolutionnels. Ces derniers sont un peu plus complexes que les entrelaceurs blocs. Toutefois, nous avons vu qu'une augmentation de la complexité n'offrait pas obligatoirement un gain dans les performances. Même si les entrelaceurs convolutionnels permettent un grand choix de paramètres, cela n'améliore pas la BER. Dans le chapitre suivant, nous allons maintenant nous intéresser à un autre type d'entrelaceur qui est aussi un entrelaceur déterministe, mais développé à partir de certaines restrictions. Il s'agit des entrelaceurs bloc-hélicoïdaux.

Chapitre 7 - Entrelaceurs hélicoïdaux

Nous avons déjà étudié les entrelaceurs classiques tels que les entrelaceurs blocs et aléatoires. Néanmoins, il existe beaucoup d'autres entrelaceurs. De nombreux types d'entrelaceurs ont été développés, mais n'ont pratiquement jamais fait l'objet d'une étude au sein des Codes Turbo. Nous avons effectué ce type d'étude, et les entrelaceurs non classiques que nous avons considérés furent de classe hélicoïdale. C'est en 1985 que Berlekamp et Tong ont eu l'idée d'une telle classe d'entrelaceurs. Ces derniers ont la particularité d'avoir une excellente synchronisation. Néanmoins, ils possèdent également des désavantages, à savoir une restriction sur la profondeur, ce qui est le cas pour les entrelaceurs hélicoïdaux purs. Nous verrons dans ce chapitre que ce désavantage peut être éliminé grâce aux nouveaux entrelaceurs bloc-hélicoïdaux, tout en conservant la propriété de synchronisation.

7.1 Entrelaceurs hélicoïdaux

7.1.1 Entrelaceurs de profondeur (N-1)

Le meilleur moyen d'introduire les entrelaceurs hélicoïdaux est par un exemple tout comme le fait D. Chi dans [15] et [16]. Néanmoins, nous pouvons tout de suite affirmer que ce type d'entrelaceurs revient en partie à lire les bits d'entrée, rangés dans une matrice, le long d'une diagonale. Tout d'abord, il est important de mentionner que les bits d'entrée sont écrits dans une matrice colonnes par colonnes. Deux termes sont primordiaux pour les entrelaceurs hélicoïdaux. Tout d'abord, la longueur de l'entrelaceur représente le nombre de lignes remplies avant de changer de colonnes. Dans l'exemple ci-dessous, à la figure 7.1, la longueur est 4. Ainsi, les bits sont écrits vers le bas par blocs de 4. Une fois 4 bits inscrits, il s'agit de changer de colonnes et de recommencer à écrire 4 bits, mais en commençant une ligne plus bas. Les bits 1 à 4 sont ainsi écrits dans

la première colonne en commençant par la première ligne. Par la suite, les bits 4 à 7 sont eux inscrits dans la deuxième colonne en commençant par la deuxième ligne. Ce processus est répété pour un certain nombre de colonnes qui correspond à la profondeur de l'entrelaceur. Dans notre cas, nous avons une profondeur de 3 étant donné que nous avons 3 colonnes. Par la suite, nous adopterons la notations H(N,P) pour décrire un entrelaceur de longueur N et de profondeur P.

1	*	*
2	5	*
3	6	9
4	7	10
13	8	11
14	17	12
15	18	21
16	19	22
25	20	23
26	29	24
27	30	33
28	31	34
	32	24
		36

Figure 7.1 – Matrice du processus d'entrelacement hélicoïdal H(4,3)

À la sortie, les bits sont lus ligne après ligne, produisant la séquence entrelacée suivante :

1	*	*	2	5	*	3	6	9	4	7	10	13	8	11	14	17	12	15	18	21
16	19	22	25	20	23	26	29	24	27	30	33	28	31	34	*	32	24	*	*	36

Figure 7.2 – Séquence de sortie de l'entrelaceur hélicoïdal H(4,3)

Il est important de noter que les symboles * ne sont là qu'à fin d'illustration pour bien comprendre le processus. En aucun cas ils ne sont transmis mais peuvent ou non être l'origine de délais suivant le design de l'entrelaceur.

Dans notre exemple, l'entrelaceur est de longueur 4 et de profondeur 3. Ceci équivaut à dire qu'une salve d'erreur doit être d'au moins 4 bits pour altérer un mot de code plus qu'une fois, un mot de code étant représenté par la longueur. En effet, supposons qu'une salve d'erreur se produise sur le bit 6 de la séquence de sortie. Comme nous considérons le bit 6, les autres bits du mot de code considéré sont 5, 7 et 8. Le bit suivant dans la séquence de sortie est le 9 et ne fait pas partie du mot de code. Il en est de même du bit suivant, à savoir 4. Il nous faut donc aller au bit suivant pour avoir le bit 7 et se retrouver dans le même mot de code d'origine du bit 6. Nous le constatons, la salve d'erreur doit être de longueur 4 bits pour altérer les bits 6 et 7. Ceci est une propriété intéressante des entrelaceurs hélicoïdaux. En effet, il nous est possible de contrôler la séparation des bits après entrelacement. Ainsi, si nous connaissons statistiquement la longueur moyenne des salves d'erreur d'un canal, nous pouvons en conséquence adapter l'entrelaceur pour éviter que cette salve ne frappe deux fois le même mot de code. De surplus, nous pouvons constater que les bits d'un même mot de code sont séparés exactement de quatre symboles. Il existe donc bien une synchronisation qui caractérise cet entrelaceur et qui permet d'effectuer le processus inverse aisément.

Ce que nous venons de présenter au moyen d'un exemple est généralisable. Nous avons pris le cas où la longueur est N et la profondeur N-1. Dans ce cas, le délai total de l'entrelacement est (N-2)(N-1) et la mémoire requise est N(N-1)/2. Ceci équivaut donc environ à la moitié des paramètres de mémoire et délai des entrelaceurs blocs de même profondeur. Toujours en généralisant, la synchronisation nécessaire à l'opération inverse de l'entrelacement est N(N-1). Néanmoins, un des avantages de cet entrelaceur est que, étant donnée sa symétrie, une synchronisation modulo N est suffisante. Finalement, ce type d'entrelacement est aussi connu sous le nom d'entrelacement hélicoïdal d'ordre N.

7.1.2 Entrelaceurs hélicoïdaux de profondeur autre que (N-1)

Nous venons d'introduire les entrelaceurs hélicoïdaux d'ordre N, c'est-à-dire de longueur N et de profondeur (N-1). Maintenant, nous allons nous intéresser à ces mêmes entrelaceurs, toujours de longueur N, mais de profondeur autre que (N-1). En fait, le processus est un tout petit peu différent, mais le principe reste le même. Considérons un entrelaceur dont la longueur est 8 et la profondeur 5. La matrice des bits d'entrée est construite avec le même principe qu'au paragraphe précédent, à une nuance près. En fait, dans ce cas, le $k^{ème}$ mot de code est inscrit dans la colonne (k-1)N modulo D. Ainsi, tout comme pour les entrelaceurs hélicoïdaux d'ordre N, les 8 premiers bits sont inscrits dans la première colonne (colonne 0). La différence apparaît lors de l'écriture des bits suivants. Le bit 9 est inscrit dans la quatrième colonne (il s'agit du deuxième mot de code, donc la colonne est (2-1)x8 modulo 5, c'est-à-dire la colonne 3, donc la quatrième), une ligne en dessous, tout comme l'hélicoïdal d'ordre N. Néanmoins, lorsque nous arrivons au bit 16, au lieu d'écrire sur une nouvelle ligne, nous remplissons la première ligne de la deuxième colonne. Lors de l'écriture des 8 prochains bits, nous commencerons à la troisième ligne et les deux derniers bits seront respectivement inscrits dans la première et la deuxième ligne. La colonne considérée est (3-1)x8 modulo 5, soit la colonne 1.

1	23	37	16	30
2	24	38	9	31
3	17	39	10	32
4	18	40	11	25
5	19	33	12	26
6	20	34	13	27
7	21	35	14	28
8	22	36	15	29

Figure 7.3 – Matrice du processus d'entrelacement hélicoïdal H(8,5)

À la sortie, les bits sont lus ligne après ligne, produisant la séquence entrelacée suivante :

1	23	37	16	30	2	24	38	9	31	3	17	39	10	32	4	18	40	11	25
5	19	33	12	26	6	20	34	13	27	7	21	35	14	28	8	22	36	15	29

Figure 7.4 – Séquence de sortie de l'entrelaceur hélicoïdal H(8,5)

À partir de la séquence de sortie, il est facile d'observer que chacun des bits d'un même mot de code (bits 1 à 8 pour le premier et ainsi de suite) font l'objet d'un délai de 4 bits. En effet, il y a 4 bits entre le bit 1 et le bit 2. Il en est de même entre le bit 2 et le bit 3. Ceci nous amène à apporter deux conclusions. Premièrement, la séparation de deux bits consécutifs après entrelacement est de 5. Ensuite, l'opération inverse d'entrelacement sera aisée dans le sens où une synchronisation modulo 8 est suffisante. En effet, il suffit de mettre un bit de synchronisation tous les 8 bits et le déentrelacement se fera aisément. L'opération inverse commencera par la recherche du premier bit de synchronisation. Une fois ce bit trouvé, il suffira d'aller chercher 5 bits plus loin pour avoir le deuxième et ainsi de suite. Une fois les 8 premiers bits retrouvés, la recherche du deuxième bit de synchronisation peut commencer. Une fois ce bit trouvé, nous recommençons l'opération de prendre un bit tous les 5.

Nous venons de généraliser l'entrelacement hélicoïdal. Néanmoins, nous avons vu que lors du processus d'entrelacement, le $k^{ème}$ mot de code était écrit dans la colonne (k-1)N modulo D. Nous voyons immédiatement que dans certains cas, ce processus ne pourra remplir toutes les colonnes de la matrice entrelacée. En effet, si nous prenons N = 8 et $D = 4$, tous les mots de code seront écrits dans la première colonne. Il est donc nécessaire de donner une restriction pour que l'entrelaceur hélicoïdal puisse être généré correctement. Une condition suffisante est que D doit être relativement premier avec N. Ceci a été évoqué par [38].

7.2 Entrelaceurs bloc-hélicoïdaux (BH) [15]

Nous avons déjà vu que les entrelaceurs hélicoïdaux possédaient une meilleure synchronisation, en particulier, ceux dont la profondeur est autre que (N-1) si la longueur de mot de code est N. Néanmoins, ils possèdent le désavantage de nécessiter que D et N soient premiers entre eux. Il existe également une autre restriction, à savoir que cet algorithme ne fonctionne que si un seul symbole de synchronisation est introduit entre deux mots de code. Si nous en introduisons plus, cet algorithme ne fonctionnera plus. Ces deux restrictions amènent à penser qu'une amélioration peut être apportée à cet algorithme. C'est pourquoi les entrelaceurs BH ont été introduits.

7.2.1 Notations

Tout au long de la description des entrelaceurs BH, nous allons utiliser certaines notations, en particulier pour les démonstrations mathématiques :

- m % N est le reste de la division de m par N, $0 \leq m \% N \leq N$;
- [a] est le plus grand entier inférieur ou égal à a;
- (m,n) est le plus grand commun diviseur des entiers m et n;
- N est la longueur d'un mot de code, ce dernier incluant le bit de synchronisation;
- D est la profondeur d'entrelacement, $D > 1$;
- nous définissons R comme : $R = (N-1)\% D$

7.2.2 Introduction aux entrelaceurs BH

La base des entrelaceurs BH se tient sur trois propriétés fondamentales :

i) le processus de lecture lit bien les ND symboles une et une seule fois

ii) si on prend D symboles consécutifs dans la séquence entrelacée, ils appartiennent à des mots de code différents

iii) deux symboles de synchronisation sont séparés par N symboles

Nous allons prendre comme notations r pour la ligne d'une matrice et c pour la colonne. Nous allons également utiliser le terme de processus d'écriture qui correspond à l'écriture des bits d'entrée dans la matrice d'entrelacement. Enfin, le terme de processus de lecture fera référence à la lecture des bits à la sortie de la matrice d'entrelacement, ce qui correspond à la séquence de bits de sortie. La première idée d'algorithme est la suivante.

$$\begin{cases} \text{Processus d'écriture} : \begin{cases} r = m \,\%\, N \\ c = [m / N] \end{cases} \\ \text{Processus de lecture} : \begin{cases} r = m \,\%\, N \\ c = m \,\%D \end{cases} \end{cases} \qquad (7.1)$$

En ce qui concerne le processus d'écriture, il est assez simple étant donné qu'il s'agit d'écrire les bits colonne par colonne. Nous allons illustrer le processus de lecture sur un exemple. Prenons N = 8 et D = 5. Le tableau suivant donne dans l'ordre la lecture des bits dans la matrice d'entrelacement :

Tableau 7.1 – Ordre de lecture des lignes et des colonnes pour BH(8,5)

Ligne	Colonne	Ligne	Colonne	Ligne	Colonne	Ligne	Colonne	Ligne	Colonne
0	0	0	3	0	1	0	4	0	2
1	1	1	4	1	2	1	0	1	3
2	2	2	0	2	3	2	1	2	4
3	3	3	1	3	4	3	2	3	0
4	4	4	2	4	0	4	3	4	1
5	0	5	3	5	1	5	4	5	2
6	1	6	4	6	2	6	0	6	3
7	2	7	0	7	3	7	1	7	4

De façon intuitive, il s'agit toujours de prendre les lignes et les colonnes dans l'ordre qu'elles viennent. Ceci est dû à la définition du modulo. Ce tableau illustre l'ordre des lignes et des colonnes, mais n'est pas assez schématique. À partir de la figure suivante, il est plus aisé de voir l'évolution des bits de sortie de la matrice d'entrelacement à l'aide des flèches, en commençant par la ligne 0 et la colonne 0.

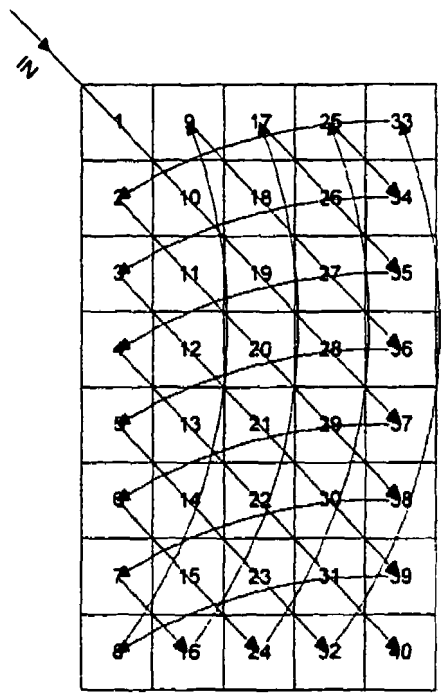


Figure 7.5 – Processus d'entrelacement BH(8,5)

La séquence de sortie de cet entrelaceur est alors (en lisant ligne par ligne):

1	10	19	28	37	6	15	24
25	34	3	12	21	30	39	8
9	18	27	36	5	14	23	32
33	2	11	20	29	38	7	16
17	26	35	4	13	22	31	40

Figure 7.6 – Séquence de sortie de l'entrelaceur BH(8,5)

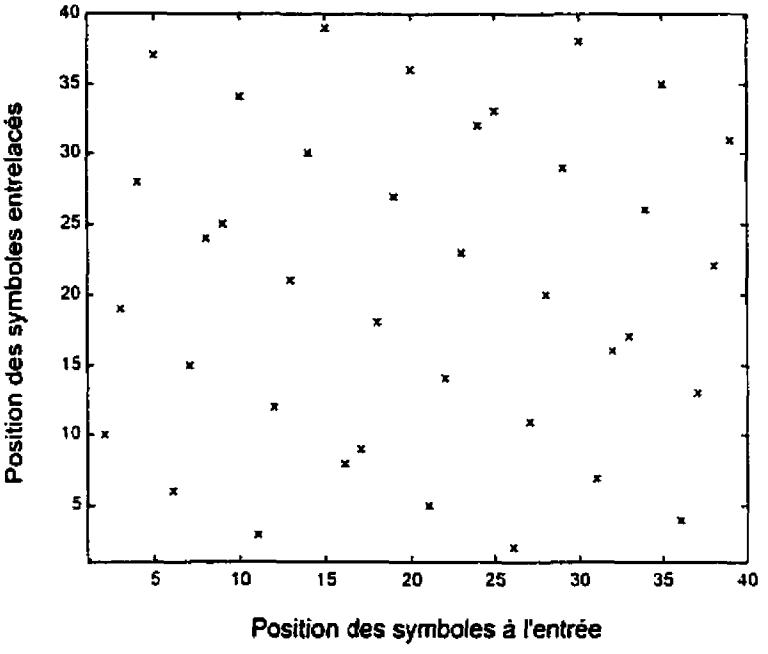


Figure 7.7 – Facteur d'étalement de l'entrelaceur BH(8,5)

Finalement, ci-dessus, nous montrons le facteur d'étalement des bits après entrelacement. Nous observons bien une certaine distance entre les bits, pour la plupart. En conséquence, une salve d'erreur ne pourra pas toucher plusieurs mots de code différents.

Nous avons pris N = 8 et D = 5, ce qui implique qu'ils sont bien premiers entre eux. De plus, à partir de la séquence de sortie de cet entrelaceur, nous pouvons affirmer que les trois propriétés des entrelaceurs BH sont remplies. Malheureusement, cet algorithme ne fonctionne pas si N et D ne sont pas premiers entre eux. Ceci implique donc qu'il existe une amélioration à apporter à cet algorithme de façon à nous débarrasser de cette contrainte. Dans les paragraphes suivants, nous allons nous concentrer à ne plus avoir de restrictions sur la longueur et la profondeur des entrelaceurs BH.

7.3 Les nouveaux entrelaceurs BH

Le but de ces entrelaceurs est de ne plus se soucier d'avoir à choisir adéquatement la longueur et la profondeur pour que l'algorithme fonctionne. Les propriétés des entrelaceurs BH doivent toutefois être remplies. Parmi les entrelaceurs BH, il existe trois catégories, dont deux sont des simplifications. Dans un premier temps, nous allons nous concentrer sur l'algorithme général de ces entrelaceurs, pour ensuite décrire les trois types. Ces derniers ont été introduits par D. Chi [15].

7.3.1 Les entrelaceurs BH – Aspect théorique

Ces entrelaceurs sont caractérisés par :

$$\begin{cases} \text{Processus d'écriture} : \begin{cases} r = m \,\%\, N \\ c = [m / N] \end{cases} \\ \text{Processus de lecture} : \begin{cases} r = m \,\%\, N \\ c = (m \,\%\, N + [m / G] + [m / N] R) \,\%\, D \end{cases} \end{cases} \tag{7.2}$$

$$\text{avec} : \begin{cases} R = (N-1) \,\%\, D \\ G = ND / (D, R) \end{cases}$$

Dans ce paragraphe, nous allons démontrer qu'en écrivant les symboles entrelacés à partir du système d'équation (7.2), les trois propriétés citées plus haut sont respectées.

7.3.1.1 Le processus de lecture lit bien les ND symboles une et une seule fois

Soit $\Phi(m) = (r,c)$. Soient deux entiers m et $\overline{m}$, il suffit alors de montrer que :

$$\begin{cases} 0 \le m < ND \\ 0 \le \overline{m} < ND \end{cases}, \text{ si } \Phi(m) = \Phi(\overline{m}) \text{ alors } m = \overline{m}.$$

Supposons que $\Phi(m) = \Phi(\overline{m})$. Alors :

$$\begin{cases} r = \overline{r} \\ c = \overline{c} \end{cases}$$

$$\begin{cases} m = qN + s & 0 \le s < N \\ \overline{m} = \overline{q}N + s & 0 \le \overline{s} < N \\ m = pG + t & 0 \le t < N \\ \overline{m} = \overline{p}G + \overline{t} & 0 \le \overline{t} < N \end{cases} \tag{7.3}$$

Alors, il vient que :

$$\begin{cases} r = \overline{r} \\ c = \overline{c} \end{cases} \Leftrightarrow$$

$$\begin{cases} m \,\%\, N = \overline{m} \,\%\, N \\ m \,\%\, N + [m/G] + [m/N]R = \overline{m} \,\%\, N + [\overline{m}/G] + [\overline{m}/N]R \pmod{D} \end{cases} \tag{7.4}$$

$$\Leftrightarrow \quad \begin{cases} s = \overline{s} \\ s + p + qR = \overline{s} + \overline{p} + \overline{q}R \pmod{D} \end{cases}$$

$$\Leftrightarrow \quad \begin{cases} s = \overline{s} \\ p + qR = \overline{p} + \overline{q}R \pmod{D} \end{cases}$$

$$\Leftrightarrow \quad \begin{cases} s = \overline{s} \\ p - \overline{p} = (q - \overline{q})R \pmod{D} \end{cases} \tag{7.5}$$

De plus, $0 \le m < DN \Leftrightarrow 0 \le \dfrac{m}{G} < \dfrac{DN}{G}$

avec $G = \frac{ND}{(D,R)}$, d'où $0 \le \frac{m}{G} < \frac{DN}{ND}(D,R)$

$$0 \le \frac{m}{G} < (D,R)$$

or, (D,R) est entier et p=[m/G], d'où, $0 \le p \le (D,R) \le R$, soit $0 \le p \le R$ ou encore

$$|p - \bar{p}| \le R \qquad (7.6)$$

En prenant en compte (7.5) et (7.6) :

$\bar{q} - q \neq 0 \Rightarrow p - \bar{p} \ge R$, ce qui est en contradiction avec $|p - \bar{p}| \le R$

Ainsi, $\begin{cases} q = \bar{q} \\ p = \bar{p} \\ s = \bar{s} \end{cases}$ soit $m = \bar{m}$

En conclusion, le processus de lecture lit les bits une et une seule fois et les lit tous.

7.3.1.2 D symboles entrelacés qui se suivent appartiennent à des mots de code différents

Pour montrer cette affirmation, il nous faut d'abord énoncer un lemme :

soit $\begin{cases} \Phi(m) = c \\ \Phi(m+1) = \bar{c} \end{cases}$, alors :

1. $(m+1) \,\%\, N \neq 0 \Rightarrow \bar{c} = c + 1 \pmod{D}$

2. $(m+1)\,\%\,N = 0 \Rightarrow \bar{c} = c \pmod D$
ou $\bar{c} = c+1 \pmod D$

1. $(m+1)\,\%\,N \neq 0 \Rightarrow \bar{c} = c+1 \pmod D$

Soit $m = Nq + (r-1) \quad 0 \leq r-1 < N$

Nous savons que $\bar{c} = r + [(m-1)/G] + qR \pmod D$

$$c = r-1 + [m/G] + qR \pmod D \qquad (7.7)$$

$$(7.7) \Rightarrow c+1-[m/G] = r+qR$$
$$\Rightarrow \bar{c} = c+1+[(m+1)/G]-[m/G]$$

Montrons alors que $[(m+1)/G] = [m/G]$

c'est-à-dire que $(m+1)\,\%\,G \neq 0$, car dans ce cas, (m+1) / G et m / G ont même quotient.

Supposons que $(m+1)\%G = 0$, avec $G = \dfrac{ND}{(D,R)}$

Alors :

$$\begin{cases} \dfrac{ND}{(D,R)} \text{ divise } (m+1) \\ N \text{ divise } (m+1) \text{ car est un entier car } (D,R) \text{ est le plus grand diviseur commun} \end{cases}$$

Cette dernière affirmation est en contradiction avec l'hypothèse que nous avons émise, à savoir $(m+1)\,\%\,N \neq 0$.

Donc, $\lceil (m+1)/G \rceil = \lceil m/G \rceil$ et $\bar{c} = c+1 \pmod D$ (7.8)

2. $(m+1)\,\%\,N = 0 \Rightarrow \begin{cases} \bar{c} = c \pmod D \\ \text{ou } \bar{c} = c+1 \pmod D \end{cases}$

Supposons $(m+1)\,\%\,N = 0$

Alors :

$$m+1 = qN$$
$$m = qN-1$$
$$m = (q-1)N + N - 1$$

donc, $c = N-1+\lceil m/G \rceil + (q-1)R \pmod D$

or, $R = (N-1)\,\%\,D \Rightarrow (N-1) = QD + R \pmod D$

donc $c = QD + R + \lceil m/G \rceil + (q-1)R \pmod D$

soit $c = \lceil m/G \rceil + qR \pmod D$ (7.9)

De plus,

$\Phi(m+1) = \bar{c} \Rightarrow \bar{c} = (m+1)\,\%\,N + \lceil (m+1)/G \rceil + \lceil (m+1)/N \rceil R$ avec $(m+1)\,\%\,N = 0$

D'où, $\bar{c} = \lceil (m+1)/G \rceil + qR \pmod D$ (7.10)

Les équations (7.9) et (7.10) impliquent alors que :

$$\begin{cases} \bar{c} = c+1 \\ \bar{c} = c \end{cases}$$

Grâce au lemme que nous venons de démontrer, il est alors possible de montrer que D symboles entrelacés qui se suivent appartiennent à des mots de code différents.

Considérons le système d'équations :

$$\begin{cases} \Phi(m) = c_0 \\ \Phi(m+1) = c_1 \\ \Phi(m+2) = c_2 \\ M \\ \Phi(m+D-1) = c_{D-1} \end{cases}$$

$c_i, 0 < i \leq D-1$, est incrémenté (mod D) à moins qu'il y ait un entier j tel que $1 < j < D-1$ et $(m+j)\%N = 0$. Or, le lemme que nous avons montré prouve que ce $(m+j)^{ème}$ symbole n'appartient pas au mot de code des D précédents.

7.3.1.2 Deux symboles de synchronisation sont séparés par N symboles

Les symboles de synchronisation sont situés à la ligne 0. Il y a N lignes. Ainsi, deux symboles de synchronisation après entrelacement sont séparés de N symboles.

7.3.2 Les entrelaceurs BH de type I

Ce type d'entrelaceur est un cas particulier des entrelaceurs BH. Dans ce cas, il s'agit de considérer que R = 0. Ceci équivaut à dire que (N – 1), c'est-à-dire la longueur du mot de code moins un bit, est divisible par la profondeur. Dans ce cas, nous allons démontrer que l'algorithme d'entrelacement est simplifié. En effet :

si $R = 0$ alors $c = m \% D$

Soit $m = qN + r \quad 0 \leq r < N$

$m = pG + t \quad 0 \leq t < N$

Nous savons que $R = 0$ avec $R = (N-1)\,\%\,D$.

Donc D divise (N-1) et alors : $G = \dfrac{DN}{(D,R)} = \dfrac{DN}{D} = N$

D'où : $\begin{cases} p = q \\ t = r \end{cases}$

Ainsi : $c = r + p + qR \pmod D$ avec $R = 0$

$$c = r + p \pmod D$$

$$c = r + q \pmod D$$

Or, D divise $q(N-1) = qN - q = m - r - q = -c$

Donc $c = m\,\%\,D$ (7.11)

Nous voyons donc que ce cas revient exactement au cas du paragraphe d'introduction sur les entrelaceurs BH. Ce type d'entrelaceur est caractérisé de type I.

7.3.3 Les entrelaceurs BH de type II

Nous venons de voir une première simplification à l'algorithme, mais une autre est également possible. Il s'agit maintenant d'émettre l'hypothèse que D et R sont premiers entre eux, donc que D est premier avec le reste de la division de (N – 1) par D. Nous justifions cette hypothèse surtout par la simplification que celle-ci amène à l'algorithme. En effet :

Si $(D,R) = 1$ alors $c = (m\,\%\,N + \lfloor m/N \rfloor R)\,\%\,D$

$$(D,R) = 1 \quad \Rightarrow \quad G = \frac{ND}{(D,R)} = ND$$

$$\Rightarrow \quad \lfloor m/G \rfloor = \lfloor m/ND \rfloor$$

$$\text{Or, } m < ND \quad \Rightarrow \quad [m/ND] = 0$$

$$\text{Donc } c = (m \,\%\, N + [m/N]R) \,\%\, D \tag{7.12}$$

Il s'agit ici des entrelaceurs de type II. Il est caractérisé par :

$$\begin{cases} \text{Processus d'écriture} : \begin{cases} r = m \,\%\, N \\ c = [m/N] \end{cases} \\ \text{Processus de lecture} : \begin{cases} r = m \,\%\, N \\ c = (m \,\%\, N + [m/N]R) \,\%\, D \end{cases} \end{cases}$$

Comme pour le paragraphe d'introduction aux entrelaceurs BH, nous allons décrire ce processus sur un exemple, à savoir le BH(10,5), et donner quelques illustrations. Le tableau suivant donne dans l'ordre la lecture des bits dans la matrice d'entrelacement :

Tableau 7.2 – Ordre de lecture des lignes et des colonnes pour BH(10,5)

Ligne	Colonne	Ligne	Colonne	Ligne	Colonne	Ligne	Colonne	Ligne	Colonne
0	0	0	4	0	3	0	2	0	1
1	1	1	0	1	4	1	3	1	2
2	2	2	1	2	0	2	4	2	3
3	3	3	2	3	1	3	0	3	4
4	4	4	3	4	2	4	1	4	0
5	0	5	4	5	3	5	2	5	1
6	1	6	0	6	4	6	3	6	2
7	2	7	1	7	0	7	4	7	3
8	3	8	2	8	1	8	0	8	4
9	4	9	3	9	2	9	1	9	0

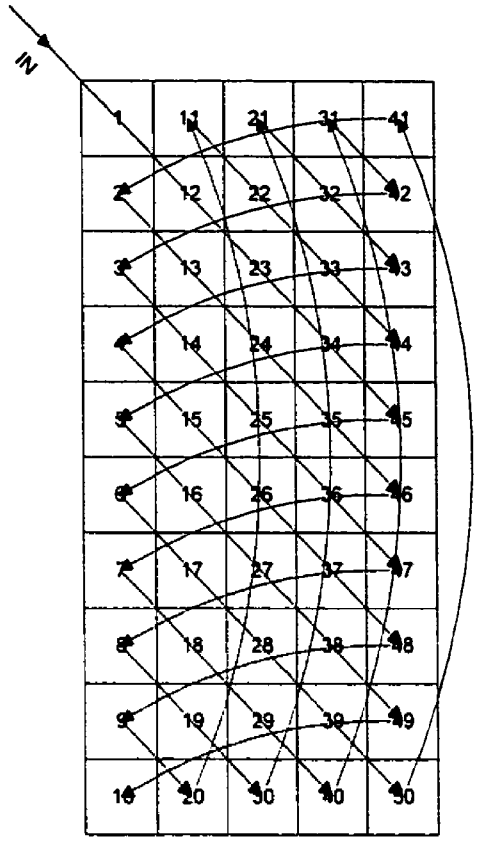


Figure 7.8 – Processus d'entrelacement BH(10,5)

À partir de la figure précédente, il nous est possible de déterminer la séquence de sortie, à savoir :

1	12	23	34	45	6	17	28	39	50
41	2	13	24	35	46	7	18	29	40
31	42	3	14	25	36	47	8	19	30
21	32	43	4	15	26	37	48	9	20
11	22	33	44	5	16	27	38	49	10

Figure 7.9 – Séquence de sortie de l'entrelaceur BH(10,5)

Enfin, le facteur d'étalement de ce type d'entrelaceur est :

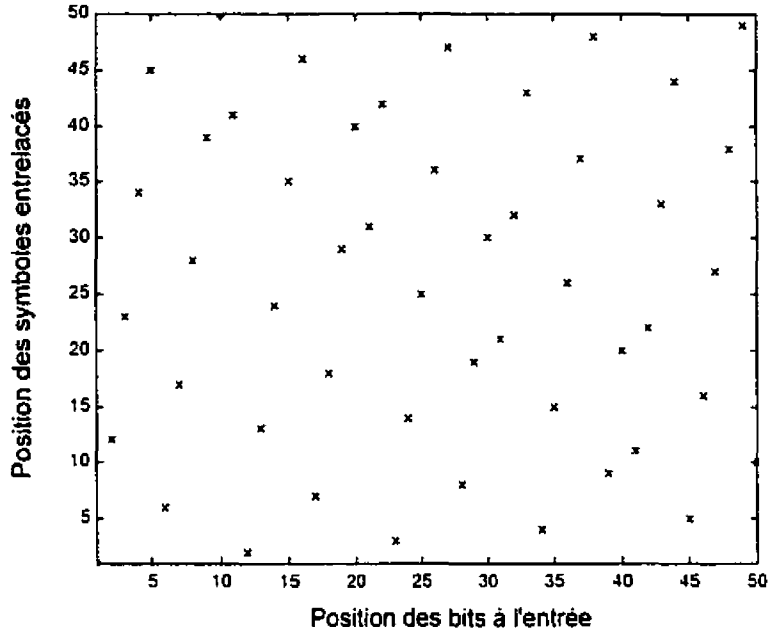


Figure 7.10 – Facteur d'étalement de l'entrelaceur BH(10,5)

Nous constatons ici que cet entrelaceur possède une bonne dispersion des bits après entrelacement. En effet, chaque bit contigu est distant de ses deux voisins de 11 bits ou bien de 9 bits. Ainsi, une salve d'erreur inférieure à 9 bits n'affectera jamais deux fois le même mot de code. Ceci peut donc s'avérer très intéressant.

7.3.4 Les entrelaceurs BH de type III

Les entrelaceurs de type III sont le cas général des entrelaceurs que nous avons décrit. L'algorithme est donc celui que nous avons déjà énoncé, à savoir :

$$\begin{cases} \text{Processus d'écriture} : \begin{cases} r = m \,\%\, N \\ c = [m / N] \end{cases} \\ \text{Processus de lecture} : \begin{cases} r = m \,\%\, N \\ c = (m \,\%\, N + [m / G] + [m / N] R) \,\%\, D \end{cases} \end{cases}$$

Illustrons ceci à l'aide d'un exemple. Nous prenons ici N = 10 et D = 6. La lecture des lignes et colonnes se fait selon le tableau suivant :

Tableau 7.3 – Ordre de lecture des lignes et des colonnes pour BH(10,6)

Ligne	Colonne	Ligne	Colonne	Ligne	Colonne	Ligne	Colonne	Ligne	Colonne
0	0	0	3	0	1	0	4	0	2
1	1	1	4	1	2	1	5	1	3
2	2	2	5	2	3	2	0	2	4
3	3	3	0	3	4	3	1	3	5
4	4	4	1	4	5	4	2	4	0
5	5	5	2	5	0	5	3	5	1
6	0	6	3	6	1	6	4	6	2
7	1	7	4	7	2	7	5	7	3
8	2	8	5	8	3	8	0	8	4
9	3	9	0	9	4	9	1	9	5

Ligne	Colonne
0	5
1	0
2	1
3	2
4	3
5	4
6	5
7	0
8	1
9	2

Quant aux processus d'entrelacement, à la séquence de sortie, et au facteur d'étalement, ils sont illustrés aux figures 7.11, 7.12 et 7.13.

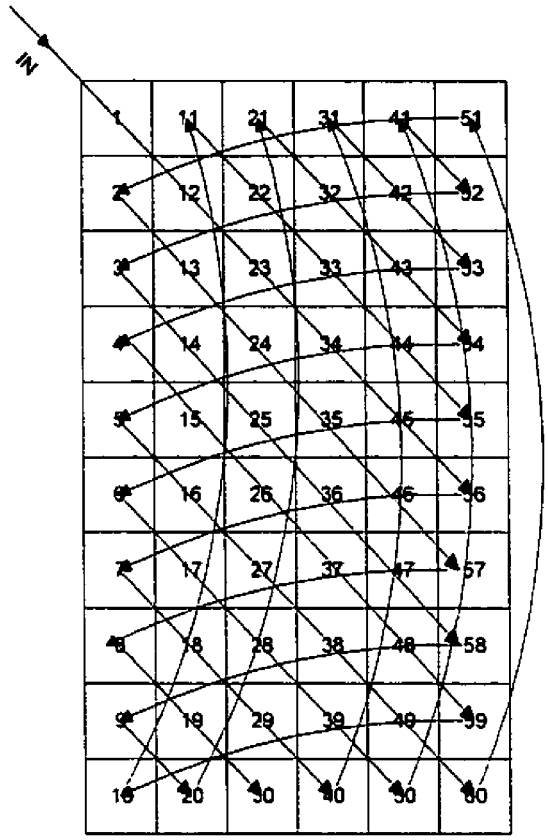


Figure 7.11 – Processus d'entrelacement BH(10,6)

1	12	23	34	45	56	7	18	29	40
31	42	53	4	15	26	37	48	59	10
11	22	33	44	55	6	17	28	39	50
41	52	3	14	25	36	47	58	9	20
21	32	43	54	5	16	27	38	49	60
51	2	13	24	35	46	57	8	19	30

Figure 7.12 – Séquence de sortie de l'entrelaceur BH(10,6)

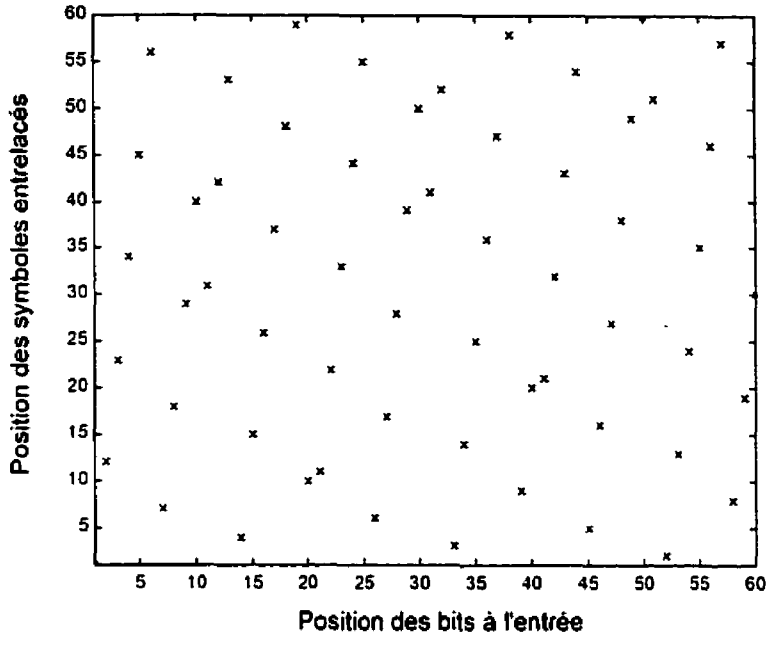


Figure 7.13 – Facteur d'étalement de l'entrelaceur BH(10,6)

Dans cet exemple, nous notons qu'un tel entrelaceur peut ne pas être efficace. En effet, après entrelacement, les bits 20 et 21 par exemple, sont encore l'un à côté de l'autre. Ainsi, si une erreur frappe ces deux bits, il se pourrait qu'une erreur frappe ces deux même bits ou l'un des deux après entrelacement, ce qui rendrait le travail du décodeur assez difficile. Il est donc bon de noter qu'un tel entrelaceur est à choisir avec précaution. Il se peut qu'il soit très efficace en ce qui concerne la dispersion des bits, comme l'exemple du BH(10,5), mais le contraire est tout à fait possible, comme ce dernier exemple.

7.4 Résultats

Comme aux chapitres précédents, nous avons effectué de nombreuses simulations pour évaluer les entrelaceurs BH grâce à notre simulateur. Nous avons utilisé un codeur turbo de taux global 1/3, de longueur de contrainte 3, dans un canal AWGN. Pour une taille de

bloc donnée, il existe de nombreux entrelaceurs BH, en fonction du nombre de lignes et du nombre de colonnes que nous utilisons. Par exemple, pour des blocs de 200 bits, nous avons huit possibilités intéressantes, à savoir 4 x 50, 5 x 40, 8 x 25, 10 x 20, 20 x 10, 25 x 8, 40 x 5 et 50 x 4. Dans les prochains paragraphes, nous allons donc nous intéresser aux entrelaceurs BH et allons déterminer quelle est la situation idéale pour de meilleures performances.

7.4.1 Entrelaceurs de longueur 200 bits

Nous avons utilisé ici des tailles de blocs de 200 bits pour étudier les performances en fonction du nombre de lignes et du nombre de colonnes. Le nombre 200 offre plus de possibilités que 196. Toutefois, lorsque nous ferons la comparaison des entrelaceurs, nous utiliserons 196 bits. Les performances sont équivalentes étant donné que la taille est minime.

Vu qu'il y a huit possibilités intéressantes, il serait assez long et pénible de commencer notre étude par le tracé de chacun des facteurs d'étalement. Nous avons donc effectué une étude un peu différente qui consiste à d'abord déterminer le meilleur des entrelaceurs à l'aide des performances comparées. Ensuite, nous justifierons ceci à partir du facteur d'étalement.

Le graphique de la figure 7.14 nous indique donc les différentes performances pour les huit entrelaceurs de 200 bits que nous avons déjà nommés. Nous n'avons pas tracé toutes les itérations, mais une seule, à savoir la quatrième. À partir de ce graphe, nous pouvons affirmer que deux des entrelaceurs ont des performances très mauvaises. Il s'agit des 4 x 50 et 5 x 40. Il semblerait donc qu'un petit nombre de lignes ne permettent pas à l'entrelaceur de bien répartir les bits. Ceci se justifie par le fait que quatre ou cinq lignes ne permettent pas d'offrir une grande diagonale. Or, le principe de cet entrelaceur est de mélanger les bits par diagonale. Ainsi, si nous commençons la diagonale au bit 1 et que

nous avons quatre lignes, le cinquième bit sera probablement le bit 2 et il ne sera distant du bit 1 que de quatre bits.

Nous constatons ensuite que les deux prochains entrelaceurs dont les performances sont médiocres sont les 50 x 4 et 8 x 25. Ce dernier se justifie de la même façon que pour 4 x 50 et 5 x 40. Toutefois, le 50 x 4 est un cas un peu différent. En effet, si l'entrelaceur 40 x 5 donnait de mauvaises performances, nous pourrions affirmer qu'un grand nombre de lignes ne donne pas de bonnes performances. Néanmoins, il n'en est pas le cas. Pourtant, nous voyons qu'entre 40 lignes et 50, il y a une différence conséquente. Nous pouvons donc conclure que 50 lignes est mauvais alors que 40 semble bon. Il est à noter quand même que la différence des performances n'est pas énorme. Ainsi, utiliser un grand nombre de lignes n'est pas fondamentalement mauvais. En effet, nous avons vu que l'entrelacement BH s'effectuait par diagonale vers le bas. Donc, plus nous avons de lignes, plus grande est la diagonale, ce qui impliquera une plus grande séparation.

Enfin, les meilleurs entrelaceurs sont 20 x 10 et 25 x 8. Nous observons donc qu'un entrelaceur dont le nombre de lignes et le nombre de colonnes tend à faire de la matrice d'entrelacement une matrice carrée sont les meilleurs. Ceci se justifie par le fait que toutes les diagonales par lesquelles nous passons pour entrelacer les bits sont grandes dans le cas d'une matrice carrée.

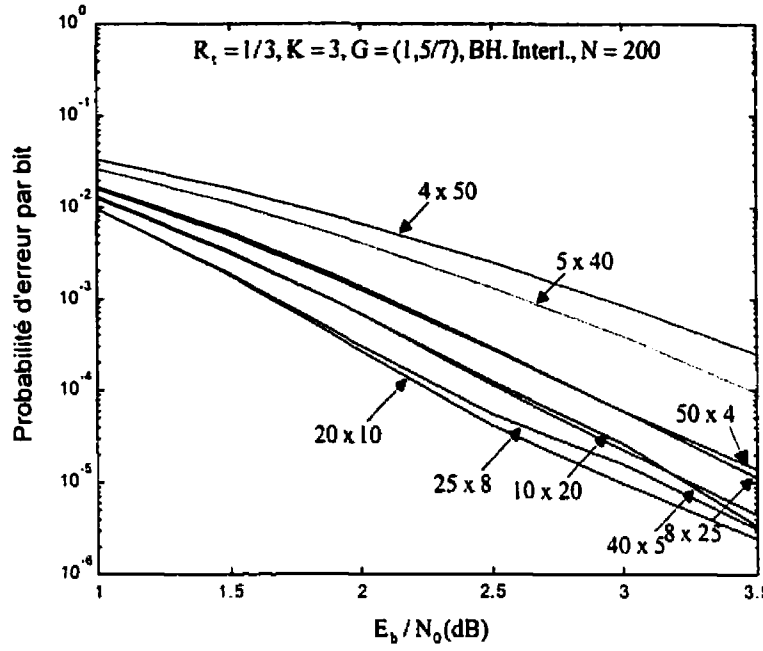


Figure 7.14 – Différentes performances des entrelaceurs BH N = 200

Nous allons maintenant passer à la justification de ce que nous venons de voir. Pour ce faire, nous avons pris le meilleur des cas et le pire. À la figure 7.15 se trouve le facteur d'entrelacement de l'entrelaceur 4 x 50. Tout de suite, nous sommes persuadés que ce dernier est mauvais étant donnée la répartition des bits. Ces derniers ne semblent même par répartis. Ils sont tout simplement distribués selon trois grandes diagonales. À l'opposé, à la figure 7.16 se trouve le facteur d'étalement de l'entrelaceur 20 x 10. Nous observons tout de suite la différence. Les bits sont beaucoup moins denses et espacés. Nous pouvons voir les diagonales suivies, mais nous observons aussi que ces diagonales sont bien espacées entre elles.

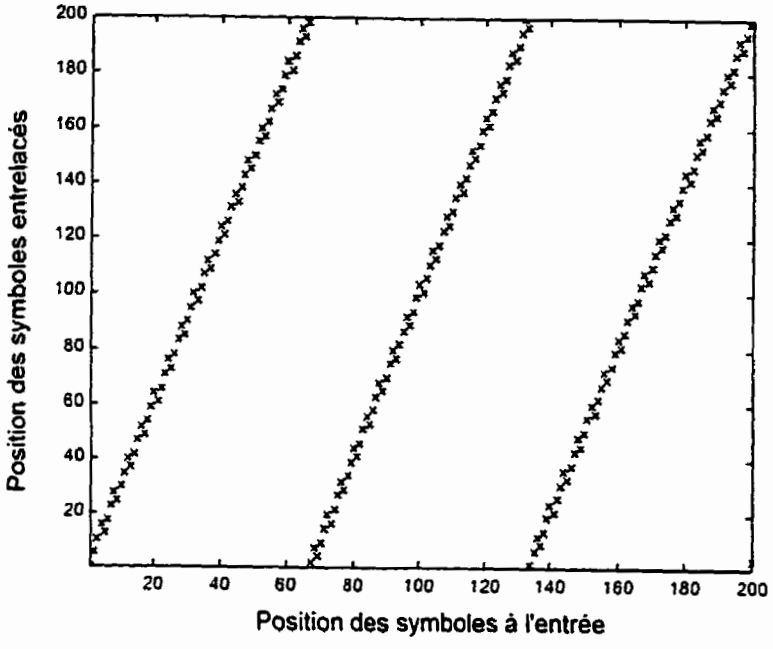


Figure 7.15 – Facteur d'étalement, entrelaceur BH 4 x 50, N = 200

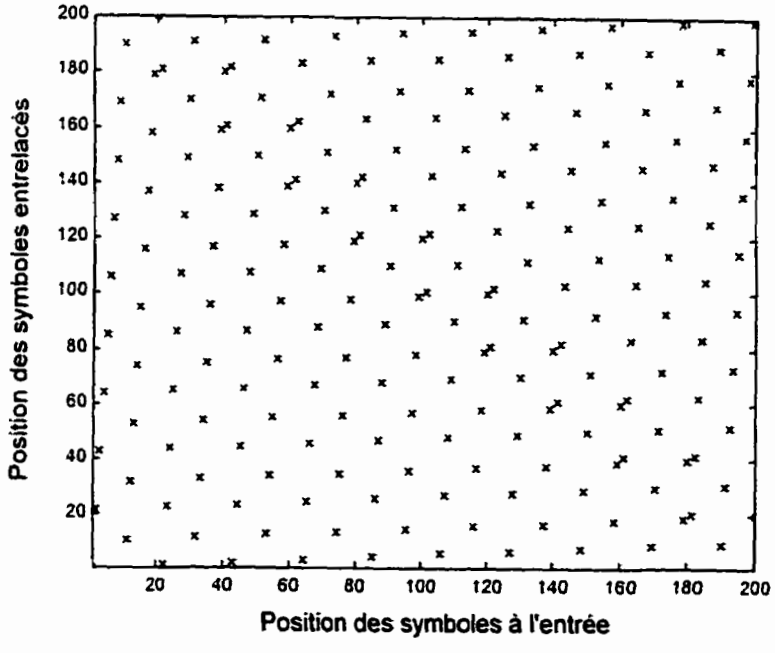


Figure 7.16 – Facteur d'étalement, entrelaceur BH 20 x 10, N = 200

Nous sommes donc maintenant sûrs que le meilleur des entrelaceurs BH est la combinaison 20 x 10. Nous avons alors tracé la courbe des performances de cet entrelaceur à la figure 7.17. Comme pour tous les entrelaceurs, nous observons de très bons gains entre les différentes itérations jusqu'à la troisième. Globalement, cet entrelaceur semble avoir de bons résultats pour une complexité moindre. Nous allons maintenant nous intéresser à des tailles de blocs plus importantes.

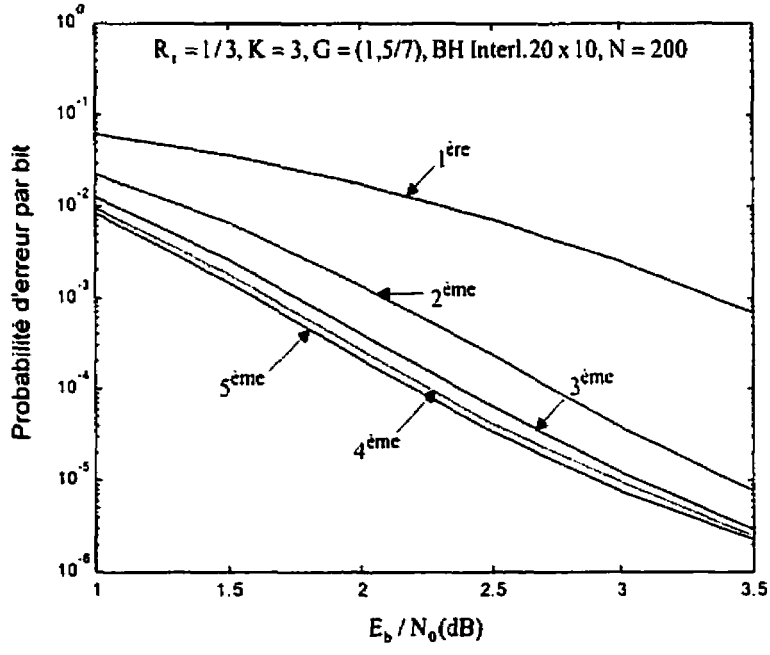


Figure 7.17 – BER, $R_t = 1/3$, K = 3, entrelaceur BH 20 x 10, canal AWGN, N = 200

7.4.2 Entrelaceurs de longueur 400 bits

Dans les mêmes conditions que précédemment, nous avons effectué nos simulations pour des blocs de 400 bits. Nous avons utilisé huit combinaisons, à savoir 8 x 50, 5 x 40, 16 x 25, 20 x 20, 25 x 16, 40 x 10, et 50 x 8. À la figure 7.18 se trouve la comparaison des performances pour ces huit entrelaceurs, toujours à la quatrième itération. Il existe une différence par rapport à des blocs de 200 bits. Autant pour ces derniers les

entrelaceurs pouvaient être séparés en trois catégories, à savoir "mauvaises performances", "performances moyennes" et "bonnes performances", autant ceci n'est plus le cas ici. En effet, nous n'avons que deux catégories.

Tout d'abord, deux de ces entrelaceurs paraissent très mauvais. Il s'agit de 8 x 50 et 5 x 40. Ceci confirme donc ce que nous avons dit pour des blocs de 200 bits. Pour un petit nombre de lignes, les entrelaceurs ne sont pas capables de séparer adéquatement les bits et il en résulte de mauvaises performances.

Enfin, tous les autres entrelaceurs possèdent des performances similaires qui ont l'air correct. Ceci tend donc à montrer que plus nous augmentons la taille des blocs, moins les paramètres de lignes et de colonnes sont capables de modifier les performances. En fait, il en est tout autre. Nous allons voir pourquoi dans l'étude du facteur d'étalement. Mentionnons toutefois que pour la suite, nous allons considérer que le meilleur de ces entrelaceurs est le 16 x 25.

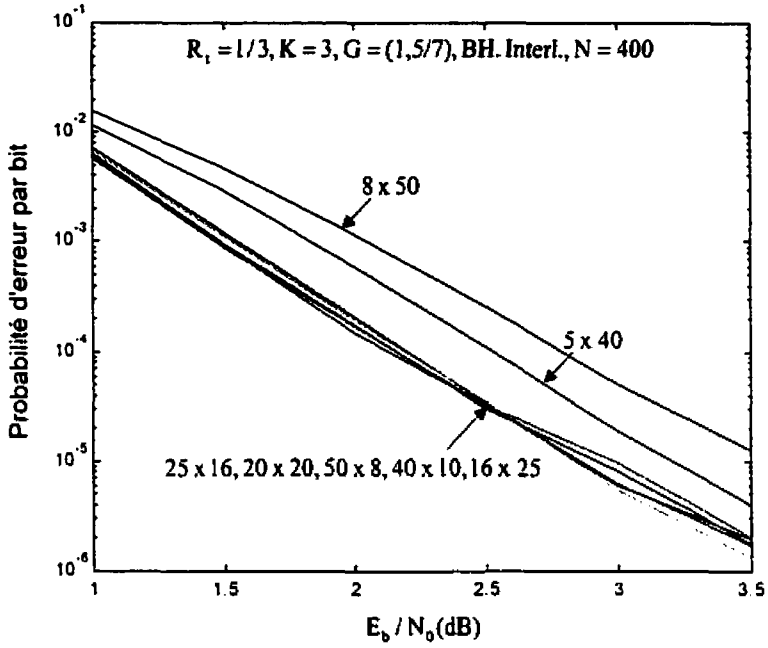


Figure 7.18 – Différentes performances des entrelaceurs BH N = 400

Nous venons de voir que ce type d'entrelaceur semble non adéquat pour mélanger les blocs de grandes tailles. Nous avons tout d'abord tracé le facteur d'étalement du pire des cas, à savoir le 8 x 50. Tout comme des tailles de 200 bits, nous observons que cet entrelaceur ne peut séparer les bits correctement. Il apparaît immédiatement les diagonales utilisées pas la matrice. Enfin, nous avons tracé le facteur d'étalement de l'entrelaceur 16 x 25 à la figure 7.20. Autant pour le cas de 200 bits la séparation semblait bonne, autant dans ce cas nous observons qu'il existe des zones non utilisées. Cet entrelaceur ne semble donc pas utiliser tout l'espace disponible pour mélanger les bits. Les diagonales utilisées sont trop évidentes. C'est pourquoi nous n'obtenons que des performances moyennes que nous avons tracées à la figure 7.21. Au paragraphe suivant, nous allons finir notre étude avec le cas de 900 bits.

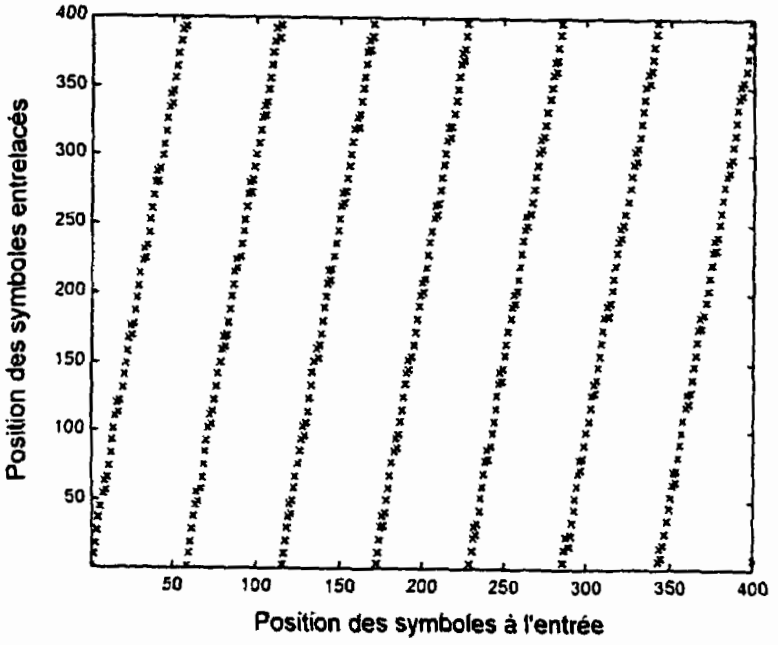


Figure 7.19 – Facteur d'étalement, entrelaceur BH 8 x 50, N = 400

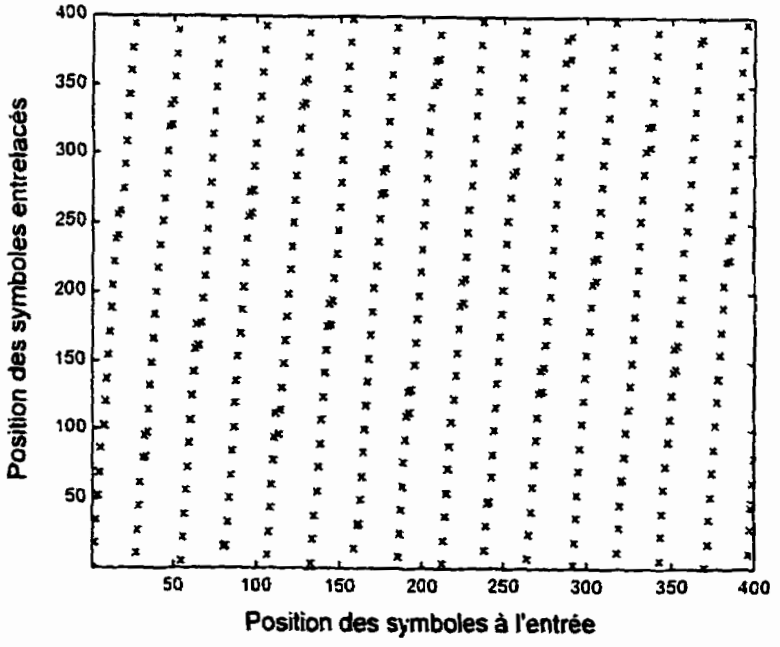


Figure 7.20 – Facteur d'étalement, entrelaceur BH 16 x 25, N = 400

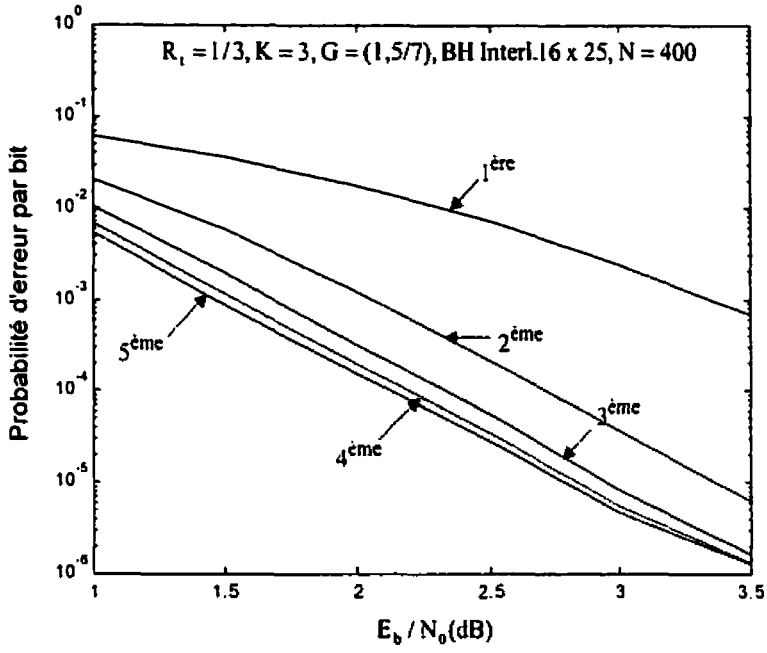


Figure 7.21 – BER, $R_t = 1/3$, $K = 3$, entrelaceur BH 16 x 25, canal AWGN, N = 400

7.4.3 Entrelaceurs de longueur 900 bits

Nous avons tracé, à la figure 7.22, différents entrelacements de type BH pour des tailles de bloc de 900 bits. Nous avons ici utilisé neuf combinaisons : 15 x 60, 18 x 50, 20 x 45, 25 x 36, 30 x 30, 36 x 25, 45 x 20, 50 x 18 et 60 x 15.

Le comportement que nous avons illustré au paragraphe précédent se confirme ici encore. En effet, nous notons qu'il n'y a plus qu'un seul de ces entrelaceurs qui semble se détacher des autres et encore, ceci est moins significatif que précédemment. En fait, tous les entrelaceurs semblent donner le même type de performances. Il semblerait donc que, quelle que soit la combinaison utilisée, l'entrelaceur BH est incapable de bien séparer les bits. Ceci se justifie aux figures suivantes, 7.23 et 7.24 où nous avons tracé les facteurs d'étalement des combinaisons 15 x 60 et 30 x 30. Il semble que la séparation des bits pour le premier est vraiment mauvaise et que celle de la deuxième est meilleure.

Toutefois, même dans le meilleur des cas, cette séparation est insuffisante pour donner de bonnes performances. Ceci se montre à la figure 7.25 où nous avons tracé les performances de l'entrelacement BH 30 x 30. Cet entrelacement, pour de grandes tailles de bloc ne semble donc pas adéquat.

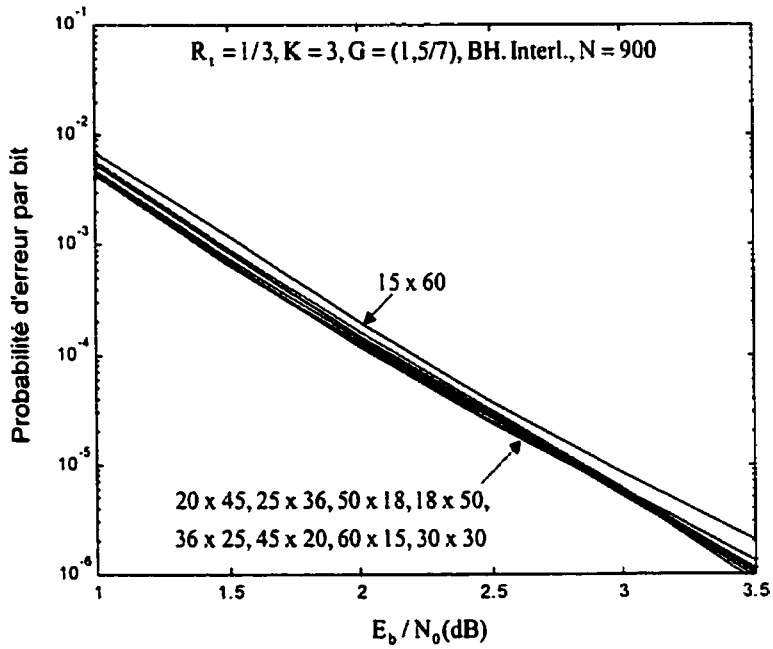


Figure 7.22 – Différentes performances des entrelaceurs BH N = 900, 4ème itération

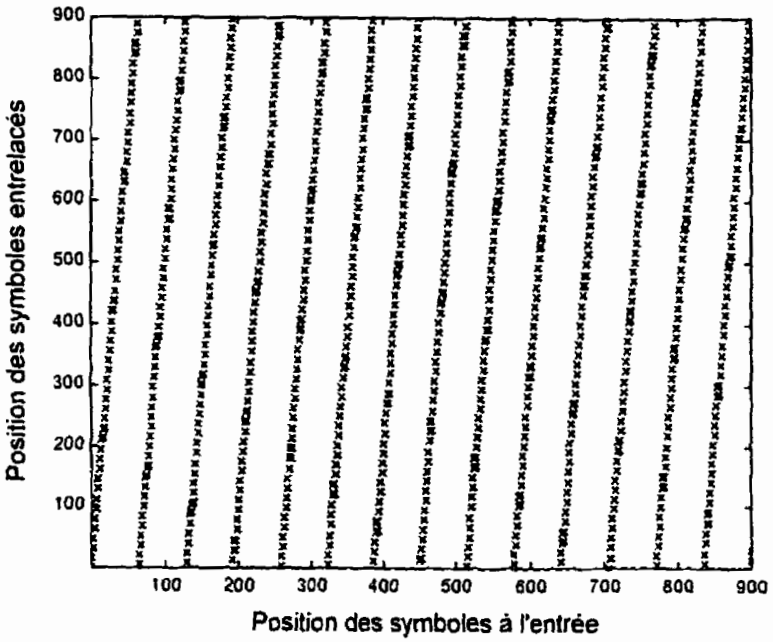


Figure 7.23 – Facteur d'étalement, entrelaceur BH 15 x 60, N = 900

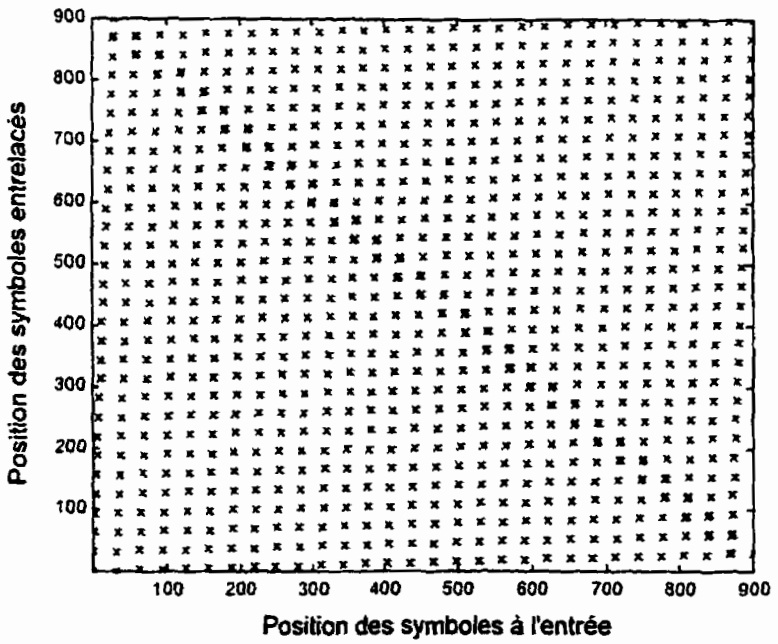


Figure 7.24 – Facteur d'étalement, entrelaceur BH 30 x 30, N = 900

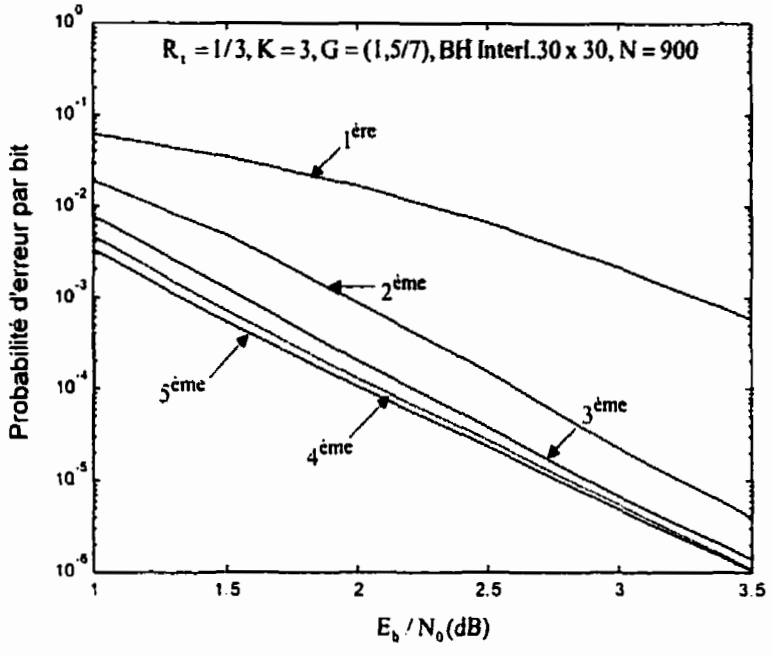


Figure 7.25 – BER, R_t = 1/3, K = 3, entrelaceur BH 30 x 30, canal AWGN, N = 900

7.5 Conclusion

Dans ce chapitre, nous avons introduit les entrelaceurs bloc-hélicoïdaux et d'en étudier les performances. Nous avons, dans un premier temps, décrit le processus d'entrelacement des entrelaceurs hélicoïdaux pour ensuite nous intéresser à ces nouveaux entrelaceurs BH. Nous avons démontré l'intérêt de ces entrelaceurs et qu'ils étaient intéressants pour de petites tailles de blocs, mais qu'ils semblaient ne plus être performants en augmentant le nombre de bits par bloc transmis.

Nous terminons ainsi notre étude de quelques uns des entrelaceurs qui existent. Dans le prochain chapitre, qui sera le sujet de notre conclusion, nous allons nous évertuer à comparer les différents entrelaceurs que nous avons étudiés dans ce mémoire. Nous serons alors en mesure de choisir un type d'entrelaceur pour chaque cas.

Chapitre 8 – Conclusion

Ce chapitre de conclusion se divise en deux parties. Dans un premier temps, nous allons effectuer une synthèse de nos résultats de façon à faire ressortir, dans chaque cas considéré, le meilleur des entrelaceurs. L'accent sera porté sur les trois cas que nous avons considérés, à savoir 196, 400 et 900 bits. Évidemment, le cas 900 bits inclut aussi des tailles de bloc supérieures. Le comportement des entrelaceurs pour des tailles supérieures est similaire au cas de 900 bits.

Dans un deuxième temps, nous allons nous concentrer sur le récapitulatif de chacun des chapitres que nous avons rédigés. Nous effectuerons une synthèse des différentes études menées ainsi que des méthodes utilisées. Enfin, nous terminerons cette conclusion par des nouvelles perspectives de recherches à considérer dans le domaine du codage correcteur d'erreurs.

8.1 Comparaison des résultats

Le mémoire que nous avons écrit a porté sur l'étude des entrelaceurs au sein des Codes Turbo. Nous en avons vu plusieurs types. Toutefois, l'étude que nous avons effectuée pour le moment était marginale. En effet, nous avons considéré chacun des entrelaceurs l'un après l'autre. Il est donc tout à fait légitime de penser à effectuer une synthèse générale sur tous les résultats que nous avons obtenus.

Nous avons donc comparé les différents entrelaceurs que nous avons étudiés. Aux figures 8.1, 8.2 et 8.3, nous avons tracé trois graphiques de comparaison des meilleurs entrelaceurs dans chacun des cas. La quatrième itération a été utilisée. À la figure 8.1, nous avons utilisé des tailles de bloc de 196 bits.

Il ressort immédiatement de ce graphique que le meilleur entrelaceur est sans aucun doute le symétrique (7). Toutefois, nous nous souvenons aussi que cet entrelaceur implique un certain délai dû au nombre d'itérations nécessaires à son implémentation. C'est pourquoi même si les autres entrelaceurs possèdent des performances relativement moindres, nous pensons que pour un design à 196 bits, ils sont à considérer. Après le symétrique (7), le meilleur des entrelaceurs est le bloc. Nous savons que ce dernier est très simple à implanter. La faible différence de performances avec le symétrique (7) ne justifie alors pas d'utiliser un algorithme complexe. Nous pensons effectivement que pour le cas 196 bits, l'utilisation de l'entrelaceur bloc est justifiée. Néanmoins, il est bon de savoir que nous pouvons avoir de meilleures performances avec le symétrique. De façon surprenante, l'entrelaceur bloc surclasse des entrelaceurs aléatoires comme les symétriques (3) ou le pseudo-aléatoire. Nous en concluons donc que pour des tailles de bloc moindres, la notion d'aléatoire n'apporte pas énormément de gain. Nous allons même jusqu'à constater que l'entrelaceur BH est meilleur que le pseudo-aléatoire.

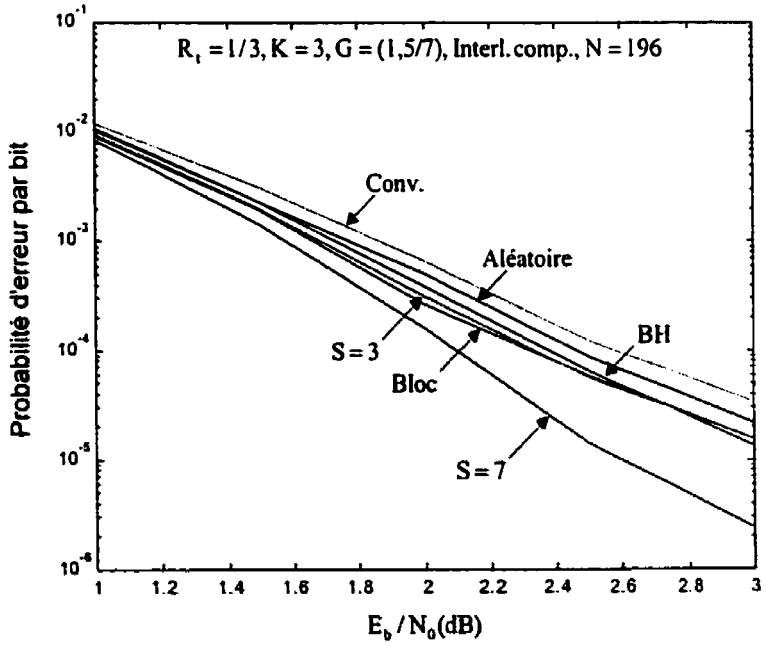


Figure 8.1 – Comparaison des entrelaceurs, N = 196, 4ème itération

Le cas 400 bits est un peu différent du cas précédent. En effet, nous constatons immédiatement une grande diversité dans les résultats. Tout d'abord, nous observons qu'un des entrelaceurs est à extraire du reste, à savoir le bloc. Ce dernier semble avoir des performances très médiocres par rapport aux autres. En effet, nous avons vu que ce dernier possédait un facteur d'étalement moyen à mesure que la taille des blocs augmentait.

Par la suite, nous constatons encore que beaucoup d'entrelaceurs possèdent des performances similaires. Il s'agit des entrelaceurs convolutionnel, BH, pseudo-aléatoire et symétrique (3). Toutefois, nous notons que ce sont les deux entrelaceurs qui ont la propriété d'aléatoire qui ont les meilleures performances. Il semblerait donc que cette propriété améliore de façon conséquente les performances à mesure que le taille des blocs augmente. Cette assertion est confirmée par l'entrelaceur symétrique (10) qui surclasse tous les entrelaceurs de loin.

Pour ce cas, l'entrelaceur symétrique (10) semble donc le meilleur choix. Toutefois, même si les performances sont moins bonnes, des entrelaceurs comme le convolutionnel et le BH sont à considérer.

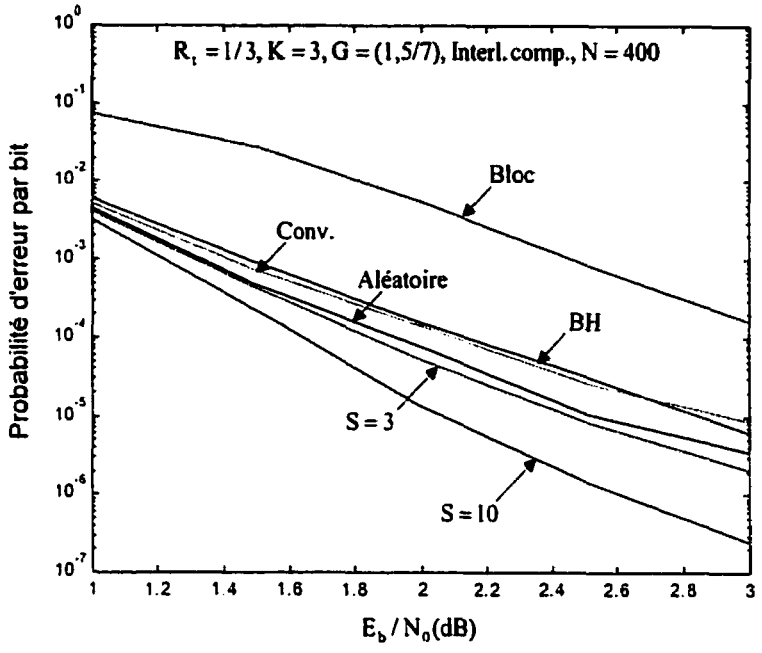


Figure 8.2 – Comparaison des entrelaceurs, N = 400, 4ème itération

Enfin, cette comparaison se termine avec le cas de 900 bits. Cette fois, les entrelaceurs sont classés en trois catégories. La première est la catégorie "médiocre". Contrairement au cas 400 bits où un seul des entrelaceurs était médiocre, nous en avons trois ici. Il s'agit des entrelaceurs bloc, convolutionnel et BH. Ainsi, pour de grandes tailles de bloc, les entrelaceurs déterministe sont inutiles.

La deuxième classe d'entrelaceurs est celle dont les performances sont bonnes et la complexité moyenne, voire faible. En effet, nous constatons que les entrelaceurs pseudo-aléatoires et symétriques (3) ont de bonnes performances sans toutefois atteindre le niveau du symétrique (15). Nous notons en outre que la différence entre le pseudo-aléatoire et le symétrique (3) est faible. Pour de grandes tailles de bloc, la notion d'aléatoire semble donc jouer un grand rôle.

Enfin, l'entrelaceur symétrique (15) possède les meilleures performances de très loin. Il semble donc adéquat d'utiliser cet entrelaceur pour de grandes tailles de bloc. En fait, à la lumière de ces résultats, nous pourrions généraliser : pour une taille de bloc N supérieure à 900 bits, le meilleur entrelaceur est le symétrique en prenant $S = \sqrt{\frac{N}{4}}$.

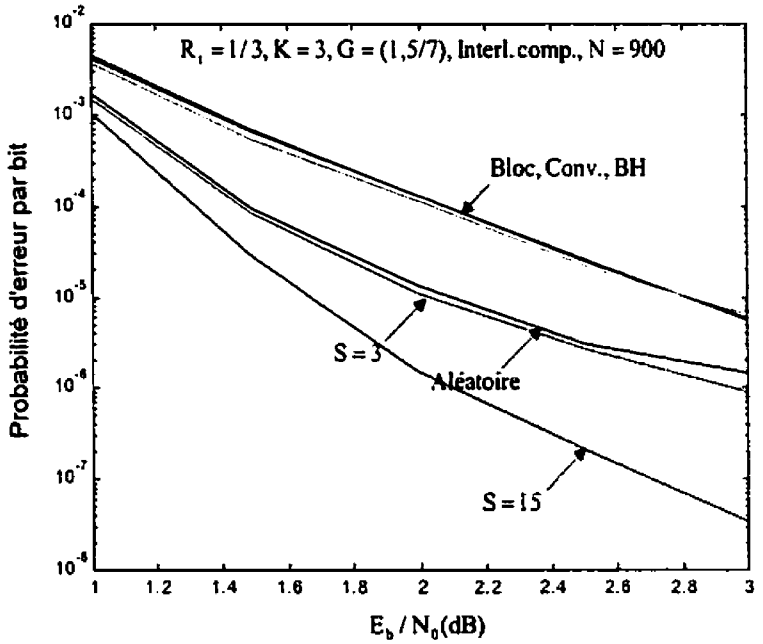


Figure 8.3 – Comparaison des entrelaceurs, N = 900, 4ème itération

Finalement, nous avons rassemblé tous ces résultats dans un tableau de façon à choisir un entrelaceur pour une BER donnée. Nous avons utilisé des BER de 10^{-3} à 10^{-6}. Dans chaque cas, nous avons mentionné l'entrelaceur dont les performances sont les meilleures. Ensuite, nous avons donné nos deux meilleurs choix en prenant en compte les performances et la complexité de l'entrelaceur. Ces résultats sont donnés au tableau ci-dessous.

Tableau 8.1 – Choix d'un entrelaceur pour une taille de bloc donnée

		Meilleur	Meilleur choix	Deuxième choix
196 bits	10^{-3}	Symétrique (7)	Bloc	Symétrique (3)
	10^{-4}	Symétrique (7)	Bloc	Symétrique (7)
	10^{-5}	Symétrique (7)	Bloc	Symétrique (7)
	10^{-6}	Symétrique (7)	Symétrique (7)	Symétrique (7)
400 bits	10^{-3}	Symétrique (10)	Aléatoire	Symétrique (3)
	10^{-4}	Symétrique (10)	Aléatoire	Symétrique (3)
	10^{-5}	Symétrique (10)	Symétrique (3)	Symétrique (10)
	10^{-6}	Symétrique (10)	Symétrique (10)	Symétrique (10)
900 bits	10^{-3}	Symétrique (15)	Aléatoire	Symétrique (3)
	10^{-4}	Symétrique (15)	Aléatoire	Symétrique (3)
	10^{-5}	Symétrique (15)	Symétrique (15)	Symétrique (15)
	10^{-6}	Symétrique (15)	Symétrique (15)	Symétrique (15)

8.2 Conclusion et perspectives de recherche

Dans ce mémoire nous nous sommes intéressés à l'étude des Codes Turbo et en particulier au comportement des entrelaceurs. Les Codes Turbo font partie de la toute dernière génération des codes correcteurs d'erreur et permettent d'approcher la limite de Shannon de très près. Dans un premier temps, nous avons concentré notre étude sur des généralités. En effet, au chapitre 2, l'effort a été porté sur l'étude générale des systèmes de transmission. Dans un premier temps, un système de communication a été présenté, pour ensuite passer au codage convolutionnel. Les codeurs convolutionnels récursifs et non récursifs ont été étudiés. Par la suite, la concaténation parallèle, à la base des Codes Turbo fut introduite pour finir par des généralités sur la modulation et les canaux de transmission.

Une fois les bases établies, il s'agissait de décrire les différentes parties qui composent les Codes Turbo. Nous avons décidé de commencer par un élément très important, à savoir l'algorithme de décodage. Il en existe plusieurs, mais nous nous sommes servis de l'algorithme MAP. L'étude qui a été menée fut très théorique et plusieurs notions furent

introduites, comme les métriques de branche. À la fin du chapitre, un récapitulatif des différentes étapes nécessaires pour générer l'algorithme a été dressé.

Au chapitre 4, nous avons décrit le décodage turbo. La notion de décodage itératif fut introduite au préalable. Ensuite, nous avons décrit en détail le processus de décodage turbo et avons présenté le schéma général d'un décodeur turbo. Ce n'est qu'à la fin de ce chapitre que nous avons étudié les entrelaceurs. Ces derniers permettent à l'information d'être décorrélée et donc aux différents codeurs et décodeurs de ne pas voir la même séquence de bits. Nous avons ainsi pu donner des paramètres primordiaux au développement d'entrelaceurs, comme le délai, la mémoire requise et le facteur d'étalement.

L'étude des performances des entrelaceurs a commencé au chapitre 5 avec les entrelaceurs pseudo-aléatoires. Les premiers étudiés furent les aléatoires pures. Ces derniers nous ont permis de décrire des résultats généraux des Codes Turbo et ainsi de voir l'effet du décodage itératif. Dans la deuxième partie de ce chapitre, nous avons introduit la notion de symétrie qui permet de rassembler en un processus l'entrelacement et le déentrelacement. Enfin, l'intérêt a été porté sur les entrelaceurs symétriques S dont les deux propriétés sont la symétrie et la séparation minimale des bits après entrelacement. Nous avons ainsi pu observer la qualité de ces entrelaceurs.

Le chapitre 6 a porté sur des entrelaceurs plus classiques et moins complexes. Nous avons d'abord porté de l'intérêt sur les entrelaceurs bloc qui sont déterministes. Nous avons pu ainsi comprendre leur processus et constater leur efficacité pour des petites tailles de blocs. Nous avons également pu constater que les performances étaient réduites à mesure que nous augmentions la taille des blocs. Par la suite, une longue étude fut portée sur les entrelaceurs convolutionnels. Une des caractéristiques de ces entrelaceurs est qu'ils sont multiplexeurs. Ils induisent un délai à la transmission. En étudiant leurs

performances, nous avons pu constater qu'ils ne semblaient pas être efficaces, quelle que soit la taille des blocs.

Enfin, le dernier chapitre a porté sur les entrelaceurs hélicoïdaux. Tout d'abord, il fut question des entrelaceurs hélicoïdaux purs pour fins d'introduction. Par la suite, l'intérêt fut porté sur les tous nouveaux entrelaceurs BH. Les performances de ces derniers sont assez bonnes et sont accompagnées d'une complexité moindre.

Autant nous avons tenté d'être le plus complet possible dans le domaine des entrelaceurs pour les Codes Turbo, autant il reste des perspectives de recherche. Voici une liste non exhaustive de recherches intéressantes sur les Codes Turbo :

- étude d'un entrelaceur qui utilise l'information extrinsèque pour générer les vecteurs de permutation;
- étendre l'étude des entrelaceurs à des longueurs de contrainte supérieures à 3 pour confirmer les résultats obtenus dans ce mémoire;
- étudier des Codes Turbo dont le nombre de codeurs est supérieur à deux;
- effectuer une étude sur la nécessité d'une queue ou non;
- comprendre l'influence de l'information extrinsèque sur les performances des Codes Turbo.

Bibliographie

[1] Bahl, L., Cocke, J., Jelinel, F., Raviv, J., "Optimal decoding of linear codes for minimizing symbol error rate", IEEE Int. Symp. Inform. Theory, p90, Asilomar, CA, 1972.

[2] Barbulescu, A., Iterative decoding of Turbo Codes and other concatenated codes, Dissertation Thesis, University of South Australia, August 1995.

[3] McAdam, P. L., Welch, L. R., Weber, C. L., "M.A.P. bit decoding of convolutionnal codes", IEEE Int. Symp. Inform. Theory, p91, Asilomar, CA, 1972.

[4] Bhargava, V., Haccoun, D., Digital Communicatoins By Satellite. New York : John Wiley, 1981.

[5] Benedetto, S., and Montorsi, G. "Unveiling Turbo Codes : Some Results on Parallel Concatenated Coding Schemes", IEEE Trans. Inform. Theory, vol. 43, pp. 409-428, Mar. 1996.

[6] Benedetto, S., Montorsi, G., Divsalar, D., and Pollora, F., "Soft-Output Decoding Algorithms in Iterative Decoding of Turbo Codes", TDA Progress Report 42-124, February 1996.

[7] Berlekamp, E. R., Po, T., "Improved Interleavers for Digital Communications", US Patent Number 4559625, Dec. 17, 1985.

[8] Berrou, C., Glavieux, A, et Thitimajshima, P., "Near Shannon limit error correcting coding and decoding : Turbo Codes", International Conference on Communications (ICC 93), pp. 1064-1070, Genève, Suisse, 1993.

[9] Berrou, C., Glavieux, A., "Near optimum error-correcting coding and decoding : Turbo Codes", IEEE Trans. Commun., vol 44, pp. 1261-1271, 1996.

[10] Belile, J., Algorithmes et architectures de décodeurs séquentiels, Mémoire de Maîtrise, École Polytechnique de Montréal, 1990.

[11] Belile, J., Décodage sous-optimal des codes convolutionnels et applications, Thèse de Doctorat, École Polytechnique de Montréal, 1993.

[12] Bouzouita, N., "Sur le décodage itératif des Codes Turbo", mémoire de maîtrise , École Polytechnique de Montréal, juin 1997.

[13] Chan, F. et Haccoun, D., "Adaptive decoding of convolutional codes", Proc. Canadian Conf. on Electrical and Computer Engineering, 69.4.1-69.4.4, Québec, Canada, 1991.

[14] Chan, F., Haccoun, D., "Adaptive decoding of convolutionnal codes", Proc. Canadien Conf. on Electrical and computer engineering, 69.4.1-69.4.4, Québec, Canada, 1991.

[15] Chi, D. T., "A new block helical interleaver", Kodak Berkeley Research, Berkeley, USA.

[16] Chi, D. T., "Helical Interleavers", Proc. of the International Telemetry Conference, vol. XXIV, pp. 465-472, Oct. 1990.

[17] Divsalar, D., and Pollora, F., "Turbo Codes for Deep-Space Communications", TDA Progress Report 42-120, Februrary 1995.

[18] Elias, P. "Error-free coding", IRE Transaction on Information Theory, vol. PGIT-4, pp. 29-37, 1954

[19] Elias, P., "Coding for Noisy channels", IRE Conv. Record, Part 4, pp. 37-47, 1955.

[20] Forney, G.D., Jr., "Concatenated codes", Cambridge : MIT Press, 1966

[21] Forney, G. D., "Convolutional codes : algebraic structure", IEEE Transf. Inform. Theory, vol. IT-16, pp. 720-738, 1970.

[22] Hagenauer, J., "Iterative decoding of binary block and convolutional codes", IEEE Transf. Inform. Theory, vol. 42, pp. 429-445, March 1996.

[23] Hagenauer, J., "Source-controlled channel decoding", IEEE Trans. Commun., vol 43, pp.2449-2457, September 1995.

[24] Hamming, R.W., "Error Detecting and Error Correcting Codes", Bell System Technical Journal, vol. 27, pp. 147-161, April 1950.

[25] Hagenauer, J., "Turbo principle: Tutorial introduction and state of the art", Proceedings of the Int. Symp. on Turbo Codes and related topics, Brest, France, 3-5 Sept. 1997.

[26] Heegard, C., Wicker, S. B., Turbo Coding, Kluwer Academic Publishers, 1999.

[27] Ho, M. S. C., Pietrobon, S. S., and Giles, T., "Improving the constituent codes of turbo encoders", IEEE Global Telecommun. Conf., vol. 6, pp. 3525-3529, Sydney, Australia, Nov. 1998.

[28] Ho, M. S. C., Pietrobon, S. S., Giles, T. "Interleavers for punctured Turbo Codes", Institute for Telecommunications Research, University of South australia

[29] Hokfelt, J., Edfors, O., and Maseng, T., "Interleaver design for turbo codes based on the performance of Iterative decoding", Lund University, Sweden, August 1998.

[30] Hokfelt, J., Edfors, O., Maseng, T., "Assessing interleaver suitability for Turbo Codes", Dept. of Applied Electronics, Lund University, Sweden.

[31] Imai, H. et Hirakawa, S, "A new multilevel coding method using error-correcting codes", IEEE Transaction on Information Theory, vol IT-23, pp. 371-377, 1977.

[32] Jung, P., and NaBan, M., "Performance evaluation of Turbo Codes for short frame transmission systems", Electronics Letters, vol. 30, pp. 111-112, Jan. 1994.

[33] Jung, P., and NaBan, M., "Comprehensive comparison of Turbo Code decoders", IEEE Vehicular Tech. Conf., pp. 645-649, 1995.

[34] Lin, L., and Cheng, R. S., "On the tail effect of SOVA-based decoding for Turbo Codes", Proc. GLOBECOM 97, Phoenix, AZ, USA, pp.644-648, November 1997.

[35] Madarel, F., "Simulation and optimisation of the turbo decoding algorithm", Final Year Project Report, University of South australia, Nov. 1996.

[36] McAdam, P. L., Welch, L. R., Weber, C. L., "M.A.P. bit decoding of convolutionnal codes", IEEE Int. Symp. Inform. Theory, p91, Asilomar, CA, 1972.

[37] Mehrotra, A., Cellular Radio Performance Engineering. Artech House, Inc., 1994.

[38] Pietrobon, S. S., "Implementation and performance of a turbo/MAP decoder", Int.J. Satellite Commun., Feb. 1997.

[39] Pietrobon, S. S., "Efficient implementation of continuous MAP decoders and a synchronization technique for turbo decoders", Int. Symp. on Inform. Theory and its applications, pp. 586-589, Victoria, BC, Canada, Sept. 1996.

[40] Reed, M., and Pietrobon, S., "Turbo Code termination schemes and a novel alternative for short frames", PIMRC 96, Taipei, Taiwan, Oct. 1996.

[41] Reed, M., and Asenstofer, J., "A novel variance estimator for turbo-code decoding", International Conference on Telecommunications, pp. 173-178, April 1997.

[42] Reed, M., and Asenstofer, J., "Numerical Analysis of the Maximum A Posteriori Algorithm", Allerton Conf. on Commun., Control, and Computing, Monticello, USA, Sep-Oct. 1997.

[43] Robertson, P., "Illuminating the structure of decoders for parallel concatenated recursive systematic (turbo) codes", IEEE Globecom conf., pp. 1298-1303, San Francisco, USA, 1994.

[44] Robertson, P., Hoeher, P., and Villebrun, E., "Optimal and sub-optimal Maximum a Posteriori algorithms suitable for turbo decoding", European Transactions on Telecommunications, ETT vol. 8, Mar-Apr. 1997.

[45] Robertson, P., Hoeher, P. and Villebrun, E., "Optimal and sub-optimal maximum a Posteriori algorithms suitable for turbo decoding", European Transactions on Telecommunications, ETT vol. 8, March-April 1997.

[46] Proakis, J.G., Digital Communication. Third edition, Mc Graw-Hill, 1995.

[47] Robertson, P., Hoeher, P. and Villebrun, E., "Optimal and sub-optimal maximum a Posteriori algorithms suitable for turbo decoding", European Transactions on Telecommunications, ETT vol. 8, March-April 1997.

[48] Shannon, C. E., "A mathematical theory of communication", Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October 1948.

[49] Shannon, C. E., "Communication Theory of Secrecy Systems", Bell System Technical Journal, 1948.

[50] Sklar, B., Digital Communications, Fundamentals and Applications. Englewood Cliffs : Prentice Hall, 1988.

[51] Sklar, B., "A primer on turbo code concepts", IEEE Comm. Magazine, pp. 94-102, Dec. 1997.

[52] Wicker, S.B., Error Control Systems for Digital Communications and storage. Englewood Cliffs : Prentice Hall, 1995.

[53] Zhang, L., "On the MAP algorithm, iterative decoding and turbo codes", in DSP for Communications Systems, Forth International Symposium, Perth, Sept. 1996.